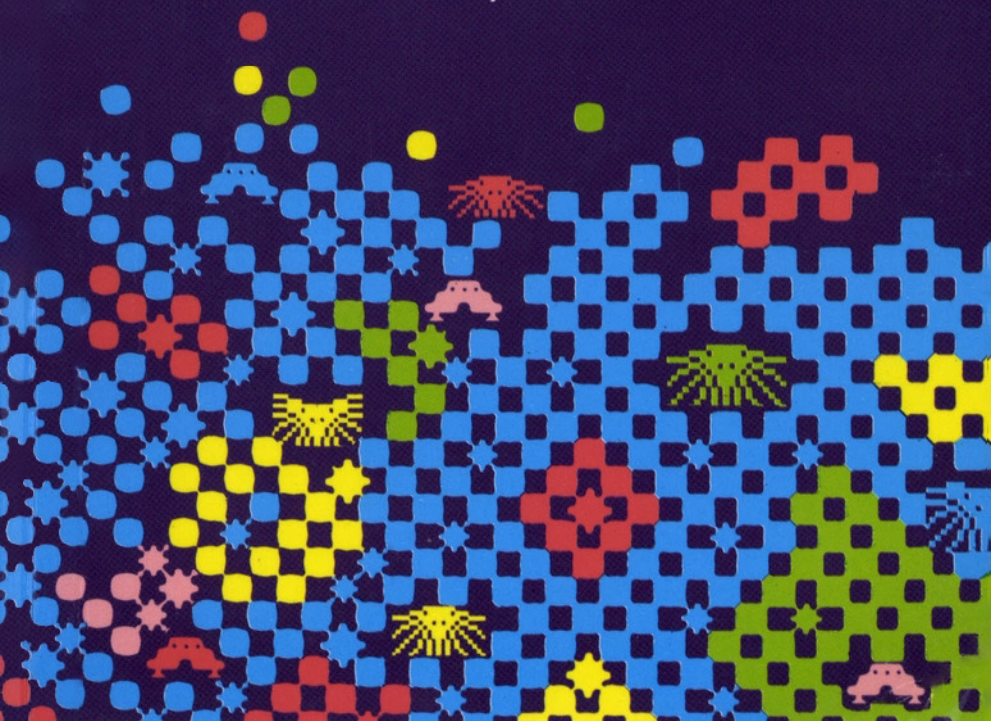
PITMAN'S FIRST BOOK OF

AMSTRAD

G·A·M·E·S

Hugh Cameron
Danny Olesh



PITMAN'S FIRST BOOK OF

AMSTRAD

G · A · M · E · S

PITMAN'S FIRST BOOK OF

AMSTRAD

G · A · M · E · S

Hugh Cameron
Danny Olesh

Pitman
Melbourne London Toronto Boston Wellington

First published 1985

Pitman Publishing Pty Ltd
(Incorporated in Victoria)

158 Bouverie Street
Carlton
Victoria 3053

Level 12
Town Hall House
452-462 Kent Street
Sydney
New South Wales 2000

9th Floor
National Bank Building
420 George Street
Brisbane
Queensland 4000

National Library of Australia
Cataloguing in Publication data

Cameron, Hugh, 1960–
Pitman's first book of Amstrad games.

ISBN 0 85896 271 3.

1. Computer games. 2. Amstrad computer. I. Olesh, Danny, 1964–. II. Title.

794.8'2

Text set in Century Old Style
by Davey Graphics (Aust) Pty Ltd

Printed in Australia
by Brown Prior Anderson Pty Ltd

Designed by Sue Veitch

Cover design by Sue Veitch

Associated companies

Pitman Publishing Ltd
London

Copp Clark Pitman
Toronto

Pitman Publishing Inc
Boston, Mass

Pitman Publishing New Zealand Ltd
Wellington

Contents

Acknowledgments

The authors would like to thank Robert Donovan of Gametronics in Melbourne for his assistance in providing some of the equipment used in writing this book. Special thanks are also due to Pieter van Wessem and Julian Tol.

A very special thanks to Sally Cameron for the patience and assistance she gave us throughout the time we were writing the book.

Hugh Cameron
Danny Olesh
March 1985

1 Introduction

This book has been written not only as a series of program listings for the Amstrad home computer, but also as an explanatory guide to many of the finer points of BASIC programming.

It is generally recognised that the quickest and easiest way to learn to write programs is to type in and study other people's programs. By doing this you become familiar with how to use the various BASIC keywords and you also learn some of the techniques other people have found useful in their programming. In this book we have gone into some detail on the more advanced techniques, such as using BIT LOGIC, and have also tried to cover many of the commands which are specific to the Amstrad or are new innovations in BASIC.

The BASIC computer programming language has arguably been the single most important factor in bringing the computer into the home. BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code, and was originally introduced as a means to program a computer that was so easy to use that it did away with the need to have a specialist computer programmer on staff. The idea was to have a programming language that was so simplified that anyone could quickly and easily learn to use it.

However, by its very simplicity, BASIC does have its drawbacks, the main one being its speed. BASIC is slower than all other popular programming languages.

Over the past few years, with the home computer revolution, BASIC has changed dramatically. Computer manufacturers have strived to improve their products as users have demanded more and more power from their machines. The BASIC as used on all the popular home computers nowadays is far more advanced than the original versions, borrowing ideas from other programming languages such as FORTRAN, PASCAL and ALGOL.

Despite the typing in of listings being the simplest way to learn to program, the first-time user will still find difficulty. Everybody makes typing mistakes, especially in the early

days when you don't really have a clue what it is that you are typing. When you are reading a typewritten letter, if you see a word which is misspelled you will make an accurate guess as to what it should be, but a computer will not accept it and will tell you it has found an *error*. That is if you are lucky. Sometimes a typing mistake *will* be accepted by the computer but the end result will be nothing like what was expected.

For this reason we have adopted some simple standards. In each listing all the keywords are printed in capital letters and all the variables in lower case. The exception to this is in REM statements, where the entire line is in capitals. The REM statements all have line numbers ending in 9. They have no bearing on the actual program, but serve as labels to help you see where you are. When you switch on your computer, you will be in Mode 1, which is 40 characters across the screen. All our listings have been printed in the same mode so that what appears on your screen will be *identical* to the listing.

Errors

Most error codes are fairly self-explanatory or can be easily understood from their description in your User Manual, appendix VIII. However there are some errors which are not adequately explained or which may not prevent the program from running but which will affect the result.

Line numbers

The first part of a program where errors are likely to occur is in the line numbering. The common feature of all the different versions of BASIC is that the programs require consecutive line numbers. This tells the computer the order in which you want the operations to be performed and also gives you labels for GOTO and GOSUB operations which tell the computer to move to another part of the program.

A very common error for both newcomers and experienced programmers alike is to enter two lines with the same line number. The effect of this is that the first line is deleted from the computer's memory. To avoid this problem the Amstrad has an AUTOmatic line numbering system. By

typing AUTO and pressing ⟨ENTER⟩ before you begin, the computer will automatically begin generating line numbers starting with 10 and incrementing by 10. All the listings in this book have been written in this way. If you wish to escape from the automatic numbering system, either because you want to retype an earlier line which you now realise has been typed incorrectly, or to enter a special line number (eg a REM statement with a line number ending in 9), then simply press the ⟨ESC⟩ key. To recommence AUTO line numbering, enter AUTO followed by the line at which you wish to recommence (eg AUTO 190). With this system, if on an AUTO line number you will be typing over an existing line, then the line number will be displayed with an asterisk (eg 190*).

Reserved words

Reserved words are often a source of errors in writing programs. The term 'reserved words' refers to the words which the BASIC interpreter recognises as keywords in the BASIC (eg PRINT, ABS, UNT). It is important that you do not attempt to use these words as variables. There is always a temptation to use the most logical name for a variable (eg 'ink' for a variable to be used for changing colours), but you should always check that the name you wish to use is not a reserved word (as in that example).

Syntax

Syntax errors are probably the single most common type of error and often the hardest ones to see. Quite simply, a syntax error will occur when the BASIC interpreter inside your computer cannot make sense of the statement you have made. Here are some typical examples.

```
10 PRINT "hello
```

Missing second " to close the string.

```
10 printa
```

Missing space after the basic keyword. Not all keywords require the space to be put in, but it is easier to put a space after every keyword. When you type in programs in lower case, the BASIC interpreter will convert all keywords it comes across to upper case when they are LISTed. If, when

you LIST a program, any keyword appears in lower case, you know it has not been recognised as a keyword. In the above example the interpreter has not recognised the PRINT statement, but has only seen the six characters 'printa'.

```
10 a=b/((256-16)*(34-6))/2+(PI/360)-4)
```

There are not an equal number of left and right brackets (parentheses). If you count them there are only four left brackets and five right ones.

```
10 DIM d
20 SOUND 1
```

Both these statements are examples of nonsense statements. They do not make sense to the BASIC interpreter.

DATA statements

Errors often arise when a long list of seemingly meaningless numbers or characters is being typed, as in a DATA statement. Typical errors are:

Not enough items in your DATA statement
If you are READing twelve items of data in your program and only have eleven items in your DATA statement, then you will get a DATA EXHAUSTED error message.

Missing separator
In a DATA statement if you miss a comma, eg

```
DATA 115,119,225 228
```

then the computer will only READ three items, in this example 115, 119, and 225228. On this occasion the computer will probably give an OVERFLOW error message (meaning that the number is too large for the computer), but the error may go unnoticed by the BASIC interpreter if, for example, the two items in question were 2 and 34.

Type mismatch

The computer will find a TYPE MISMATCH error if you try to perform either string functions on numbers or arithmetic functions on strings, eg

```
PRINT "hello"*6
```
and
```
PRINT LEFT$(455,1)
```

would both result in a TYPE MISMATCH error message. TYPE MISMATCH errors will also arise if you READ or INPUT information of the wrong type. For example if the line in the program says:

```
100 INPUT a$
```

and you then input a number (eg 4) then the program will BREAK, giving this error code. Similarly, if the program is READing data into a numeric variable (a) and it attempts to READ a string value, then again the program will BREAK, eg

```
100 FOR x=1 TO 4
110 READ a
120 NEXT x
900 DATA 200,100,35,hello
```

This error commonly occurs when you have typed in the letter O instead of the number 0 or vice versa. Making this particular mistake will result in either this error or a SYNTAX error message.

Debugging

When you have completed the typing in of a program you will now want to RUN it and see what you have achieved. At this point one of three things can happen:

- the program will run perfectly–very rare
- the program will run but not quite as expected–very common
- the program will start to run but will suddenly stop and tell you that there is an error of some sort–extremely common.

The usual situation is the third mentioned. When an error message appears you will begin 'debugging' the program. Debugging is the term for correcting errors and is traditionally the labour-intensive part of programming. By following the tips given in the previous section on entering the listings,

you should be able to minimise the amount of debugging required, but don't be too disappointed if after entering your program it won't run. When you can enter a program without having to then debug it you will be one of a very elite few.

Let's assume that you have RUN your program and now find yourself with the program execution stopped and an error message displayed on the screen telling you both the error and the number of the line in which it has occurred. Often the computer will also display the actual line containing the error (eg some syntax errors). In this case, by looking at the line and comparing it to the listing you can usually see the error. If the line is not displayed then the first step is to LIST the line. It is usually more beneficial to LIST several of the lines on either side of it as well, eg

```
Unexpected NEXT in 480
```

The procedure at this point would be to LIST the section of program immediately before line 480, since presumably you must have left out the FOR statement in an earlier line.

```
LIST 440-
```

might give

```
440 a=100
450 b=267
470 PRINT a*b
480 NEXT x
```

The first thing you would notice here is that line 460 appears to be missing. You would then look for line 460 in the listing. In this case line 460 might have been:

```
460 FOR x=1 TO a
```

Of course there may not be a FOR statement in line 460, in which case you would look for the missing FOR statement earlier in the program.

The third possibility is that line 480 is incorrect, which is easy to tell by looking at the program listing. However if this is the case then you not only have to type in the correct line 480, but also have to find out where you came up with the line '480 NEXT x'. This is quite a common error, and is caused by typing an incorrect line number later in the program. For example line 480 may have been:

```
480 d=a*b
```

but line 580 may have been:

```
580 NEXT x
```

and when you were entering the program you typed line 580 in as line 480 which then replaced the old line 480.

Always be wary that the line where the computer finds an error is not always the line in which the error has occurred. The most obvious example of this is when the program stops because a function has an invalid argument, eg

```
360 LOCATE #1,1,1
```

will give an error if you have not defined a WINDOW#1 earlier in the program.

Similarly, if an error occurs in a line which uses a variable as an argument of some function, and there appears to be no problem with the line itself, then the error probably lies in the value of the variable, eg

```
120 LOCATE x,y
```

If you have a line like this and you get an error here and you have checked it and it is the same as in the listing, then type in:

```
PRINT x
```

and

```
PRINT y
```

This is using the Direct Mode (ie not using line numbers) and will give you the values for x and y at the point when the program stopped. It may be that by doing this you find that x has a negative value and that this is why the program stopped. Now you will have to find out how x has come to have a negative value.

To do this you would compare all the lines in your program with those in the listing which contain statements which alter the value of x. This procedure is known as *tracing the error* and can sometimes be quite difficult. There are two BASIC techniques commonly used for this, the first one being to insert a STOP at various places within the program, eg

```
120 x=x+(x<1)-(x>1)
125 STOP
```

When you now RUN the program, it will stop executing at this point. You would then in the Direct Command Mode PRINT x to see whether or not it is still a legal value at this point. Assuming it is, you recommence execution of the program with the CONTinue command.

The other technique is to use the trace facility on the Amstrad. This is initiated with the command TRON and switched off again with TROFF. With the trace on, as the program runs, the line number of the line currently being executed is displayed on the screen. The advantage of using the trace facility is that you can see whether or not the program is performing all the required GOTO and GOSUB commands.

A last point with debugging. It is often advisable to SAVE your program immediately after you have finished typing it in and before you commence debugging it. This is in case when you run it you cannot break out of your program. After having SAVEd it you would then RUN it and commence debugging. When you have completed the debugging you would then SAVE the finished program.

Saving programs

Your User Manual does cover most of the basics of using cassette storage of programs, but one point that cannot be emphasised too strongly is the need to wind the cassette on a little to avoid trying to record over the blank leader at the start of the tape. On the Amstrad this problem is particularly notable because there is such a temptation to fully REWind a tape, type in SAVE and then press PLAY and RECord and thus SAVE your program because it will automatically start and stop the cassette. This will work only if you are using leaderless tapes. Most tapes have a short leader of clear or coloured plastic on either end of the actual recording tape, which will not record. If you are recording music onto a tape you don't really worry if you miss the first few notes, but the computer requires every single item of data that it stores to be recorded.

To avoid losing any information, all you need to do is wind the tape forward so that the brown part of the tape is showing before you SAVE your program.

Using shorter tapes (10 or 15 minute tapes) is preferable to using longer ones (such as C-60 or C-90) because the longer a tape is the more likely it is to stretch. If it stretches too much then the computer will be unable to read the information off it and you will have lost your program. Also, it is much easier to locate the program you want if you have five short tapes with four programs on each than if you have one long tape with twenty programs on it.

2 Graphics routines

One area of computer programming which gives a lot of pleasure for very little effort is Graphics. This section of the book gives you some sample listings of this type.

Once you have entered a program and run it a few times we would recommend that you then play around with it a little, changing the values of the variables used. There are six listings demonstrating different aspects of the high-resolution graphics available on the Amstrad computer.

Stringy

The Stringy program is an example of Moire graphics. Moire patterns are patterns where intersecting lines create interesting effects. In this listing four separate lines are being drawn at the same time to different parts of the screen, giving the effect of a square closing in. We chose Mode 2 to give a better effect of the Moire pattern. You can easily alter the effect by changing the value by which you are stepping in line 30.

```
9 REM stringy
10 MODE 2
20 CLS
30 FOR i=1 TO 400 STEP 4
40 PLOT 0,i:DRAW 640-i,0
50 PLOT i,0:DRAW 640,i
60 PLOT 640-i,400:DRAW 0,400-i
70 PLOT i,400:DRAW 640,400-i
80 NEXT i
```

Circle

One command that is lacking in the Amstrad's BASIC is a CIRCLE command. To get around this, there are various ways of drawing circles. The most usual one is to use the SIN and COS functions.

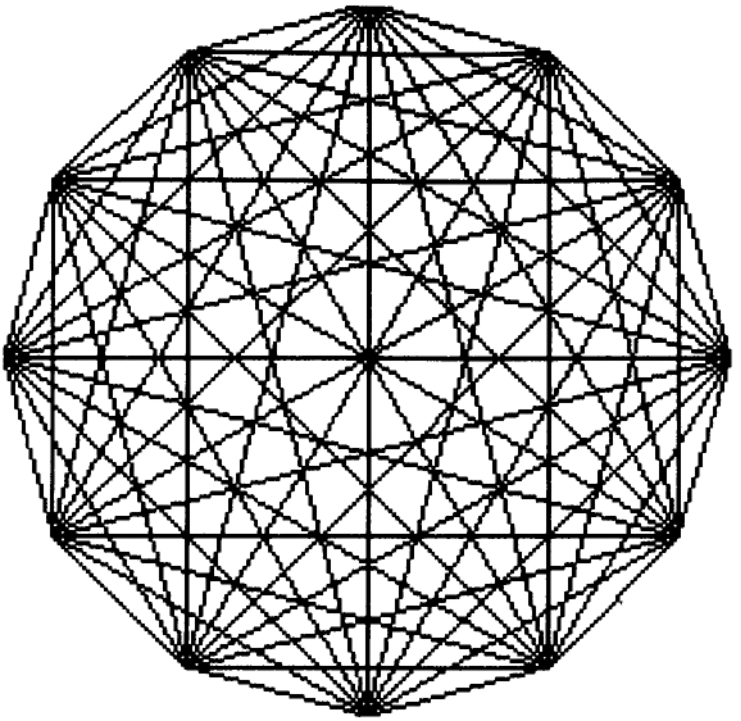
In this program we draw lines of a randomly selected radius from a randomly selected point. You can make the circle appear fuller by changing line 80. The variables x and y determine the origins for each circle and the variable r determines the radius.

```
9 REM circle
10 MODE 1
20 CLS
30 BORDER 0:INK 0,0
40 FOR i=1 TO 26
50 x=INT(RND(1)*640)
60 y=INT(RND(1)*400)
70 r=INT(RND(1)*30)
80 FOR d=1 TO 60 STEP 2
90 PLOT x,y
100 DRAWR SIN(d)*r,COS(d)*r,1
110 NEXT d
120 INK 1,i
130 NEXT i
```

Twelve sides

This program draws a twelve-sided figure but can easily be changed to draw a shape with any number of sides. The program makes use of two arrays to hold the values of the x and y positions at each corner. The values for each of the positions are calculated using the formula in line 60 which can be changed to alter the shape of the object drawn. The routine from line 90 to line 130 plots each of the points to the screen and draws lines interconnecting them. Using the DIM statements to put the values into arrays makes the program shorter and easier to understand.

```
9 REM 12sides
10 MODE 1
20 CLS
30 DIM a(12):DIM b(12)
40 FOR n=1 TO 12
50 k=n/6*PI
60 a(n)=320+200*SIN(k):b(n)=200+200*COS(
k)
70 PLOT a(n),b(n)
80 NEXT n
90 FOR n=1 TO 12
100 FOR m=1 TO 12
110 PLOT a(n),b(n):DRAW a(m),b(m)
120 NEXT m
130 NEXT n
```



Star

This is an example of using a plotting routine to produce three-dimensional effects. The program takes a while to plot all the positions, so be patient it is well worth the wait. There is a lot you can do by playing around with the formula in line 60. You may like to try:

60 PLOT 320+x*SIN(n-30),200+y*COS(n-60)

You may also like to change the stepping in line 50.

```
9 REM star
10 MODE 1
20 CLS
30 x=0
40 y=200
50 FOR n=0 TO 2*PI STEP PI/180
60 PLOT 320+x*SIN(n),200+y*COS(n)
70 NEXT n
80 x=x+10:y=y-10
```

```
9Ø IF y=-1Ø THEN STOP
1ØØ GOTO 5Ø
```

Eye

The Eye program uses the DIM statement much like the Twelve Sides program to hold the values of the points plotted.

l	Horizontal centre of the screen
j	Vertical centre of the screen
h	Colour and number of shapes drawn
n	Number of position within the array
a(n)	Horizontal position to be drawn from
b(n)	Vertical position to be drawn from
m	Number of position within the array to be drawn to
a(m)	Horizontal position to be drawn to
b(m)	Vertical position to be drawn to

The variables l and j also make the shape an ellipse rather than a circle.

```
9 REM eye
1Ø MODE 1
15 BORDER Ø:INK Ø,Ø
2Ø CLS
3Ø DIM a(36):DIM b(36)
4Ø l=32Ø: j=2ØØ
5Ø FOR h=1 TO 5
6Ø FOR n=1 TO 36
7Ø k=n/18*PI
8Ø a(n)=32Ø +l*SIN(k)
9Ø b(n)=2ØØ +j*COS(k)
1ØØ PLOT a(n),b(n)
11Ø NEXT n
12Ø FOR n=1 TO 36
13Ø m=n+12
14Ø IF m>36 THEN m=m-36
```

```
150 PLOT a(n),b(n):DRAW a(m),b(m),h
160 NEXT n
170 l=l/2:j=j/2
180 NEXT h
```

Wave

Wave is the longest of these graphics listings but by far the most satisfying and enjoyable to watch. This program makes extensive use of the DEF FuNction command. If you wish to use the same formula many times in a program it is often much easier to define a function containing the formula.

For example in this program we have defined the formula

INT(RND(1)*x)

and called it the function r(x), where x is the variable within the formula which can contain any number.

The function r can be applied to any variable, for example

FNr(y) would give the result
INT(RND(1)*y)

The function is defined by using the basic keyword 'DEF FN' and then is called up using the basic keyword 'FN'.

When you define the function you must give it a name. Then if within a program you are making use of several functions, you can call up the one you require. Although we called the function r, we could for example have called it 'random'. Make sure, however, that you don't try to call a function by a keyword, eg

DEF FN RND(x) = x + x
DEF FN PRINT (a$,j,l) = MID$(a$,j,l)

These will not work!

Variables

x,y	Determine the origin
l,m	Line drawn
u	Determines spacing between lines
v	Keeps spacing fairly even
q	Colours
g	Number of lines drawn

a,b	New origin
c,d	New line drawn
vv	Scroll the screen up

Program

Lines 150 to 180 check that you are not drawing off the screen and if you are they send the line in the opposite direction.
Line 30 ensures that the random numbers generated are truly random by making use of the internal clock.

Don't miss this one. Type it in, run it, sit back and *enjoy!*

```
9 REM wave
10 INK 0,0
20 BORDER 0
30 RANDOMIZE TIME
40 CLS
50 x=INT(RND(10)*640)
60 y=INT(RND(10)*400)
70 l=INT(RND(10)*640)
80 m=INT(RND(10)*400)
90 u=14:v=7
100 DEF FNr(x)=INT(RND(1)*x)
110 GOSUB 290
120 FOR q=1 TO 26
130 FOR g=1 TO 300
140 INK 1,q:PEN 1:PLOT x,y:DRAW l,m
150 IF x+a>640 OR x+a<0 THEN a=-a
160 IF y+b>400 OR y+b<0 THEN b=-b
170 IF l+c>640 OR l+c<0 THEN c=-c
180 IF m+d>400 OR m+c<0 THEN d=-d
190 x=x+a:y=y+b
200 l=l+c:m=m+d
210 NEXT g
220 LOCATE 1,25
230 FOR vv=1 TO 30
240 PRINT
250 NEXT
260 CLS
270 NEXT q
280 RUN
290 a=FNr(u)-v
```

```
300 b=FNr(u)-v
310 c=FNr(u)-v
320 d=FNr(u)-v
330 RETURN
```

3 Touch typing tutor

This program is a tutoring program to improve your speed and accuracy in typing. There are four levels of difficulty covering the most commonly used keys. The computer will randomly generate a string of letters which you will then copy. If at any time while typing the line you have made an error, you may recommence typing the line by pressing the ⟨ENTER⟩ key. The ⟨DELETE⟩ key will *not* work.

Level one – the Home keys (ASDFG HJKL:)

Place your left hand so that your little finger is on the A key, your ring finger is on the S key, your middle finger is on the D key and your left pointer is on the F key.

Place your right hand so that your little finger is on the key, your ring finger is on the L key, your middle finger is on the K key and your pointer is on the J key.

This is called the 'Home' position, and you should always go back to this position after you type in a letter.

The G key is pressed by moving your left pointer finger one space to the right, and the H key is pressed by moving your right pointer one space to the left.

Level two – the Home keys and the upper row (QWERT YUIOP)

The Q, W, E and R keys are all pressed with the corresponding left hand fingers. To do this from the Home position each finger moves up and to the left.

The P, O, I and U keys are all pressed with the corresponding right hand fingers. To do this from the Home position each finger moves up and to the left.

Again the T and Y keys are reached by stretching your left and right pointers that little bit further.

Level three – the Home keys, the upper row and the bottom row (ZXCVB NM,./)

The Z, X, C and V keys are all reached with the fingers of the left hand by shifting them down and to the right.

The / . , and M keys are all reached with the fingers

of the right hand by shifting them down and to the right.

The B and N keys are reached by stretching the pointers of your left and right hands.

Level four – all the previous levels plus their capitals

To print the capital of a letter typed by the left hand, depress the ⟨SHIFT⟩ key on the right side of the keyboard with the little finger of the right hand. For letters typed by the right hand, depress the left ⟨SHIFT⟩ key with the little finger of the left hand.

The space bar is pressed with the thumb of either hand.

Variables

a$(4)	An array holding the characters for the four levels of difficulty
s$(25)	A string holding the twenty-five characters used for practising
rn	A character randomly selected from a$ to be put into s$
er	The number of errors made
tm	Start time
tt	Time taken

Program

Lines 10–100 define ink colours, set up strings for a$ and define windows for screen display.

Lines 200–240 set up menu.

Lines 300–360 generate and display the string to be copied.

Lines 400–460 wait for first key input, reset errors and then start timer.

Lines 470–540 scan keyboard for input, check for errors and display keys pressed. Line 500 checks the ⟨ENTER⟩ key has been pressed; if it has you start typing the line again.

Line 600–690 calculate and print time taken and number of errors made. If there are any errors then you must type in the line again, otherwise you are returned to the menu.

Tips

Initially concentrate more on hitting the right keys with the right fingers than on worrying about time.

Try to look at the screen rather than at your fingers. A good way of making sure of this is to put a sheet of paper over your hands.

Don't miss any spaces.

A little bit of hard work will definitely pay off in the long run.

```
9 REM INITIALISE
10 CLS
20 INK 2,0:INK 3,26
30 DIM a$(4)
40 DIM s$(25)
50 a$(1)="asdf ghjkl:"
60 a$(2)="qwert yuiop"+a$(1)
70 a$(3)="zxcv bnm,./"+a$(2)
80 a$(4)=a$(3)+UPPER$(a$(3))
90 WINDOW #1,1,40,1,8
100 WINDOW #2,1,40,9,16:PAPER #2,3:PEN #
2,2
199 REM MENU
200 LOCATE 12,5:PRINT "1.  asdfg hjkl:"
210 LOCATE 12,7:PRINT "2.  qwert yuiop"
220 LOCATE 12,9:PRINT "3.  zxcvb nm,./"
230 LOCATE 12,11:PRINT "4.  UPPER CASE"
240 LOCATE 11,14:INPUT "Input selection
";a
299 REM PRINT CHARACTERS
300 CLS
310 LOCATE #1,7,5:PRINT #1,"!";
320 FOR c=1 TO 25
330 rn=INT(RND(1)*(LEN(a$(a))-1))+1
340 s$(c)=MID$(a$(a),rn,1)
350 PRINT #1,s$(c);
360 NEXT c
399 REM MAIN PROGRAM
400 CLS #2:LOCATE #2,7,5:PRINT #2,"!";
410 c=1
420 k$=INKEY$:IF k$="" THEN 420
430 SOUND 1,10,2,15,0,0,2
```

```
440 er=0
450 tm=TIME
460 GOTO 510
470 FOR c=2 TO 25
480 k$=INKEY$:IF k$="" THEN 480
490 SOUND 1,10,2,15,0,0,2
500 IF k$=CHR$(13) THEN 400
510 IF k$<>s$(c) THEN er=er+1
520 PRINT #2,k$;
530 IF c=1 THEN GOTO 470
540 NEXT c
599 REM END OF PROGRAM
600 tt=(TIME-tm)/300
610 tt=INT(tt*100)/100
620 LOCATE 1,23:PRINT "time taken ";tt;"
  errors made ";er
630 FOR s=200 TO 100 STEP -1:SOUND 1,s,0
.5,15:NEXT s:SOUND 1,s,50,15
640 PRINT:PEN 3:PRINT TAB (14);"press a
key":PEN 1
650 k$=INKEY$
660 IF k$="" THEN 650
670 LOCATE 14,25:PRINT SPACE$(12)
680 IF er>0 THEN 400
690 CLS:GOTO 200
```

4 Ski run

In this program you have to control a skier down a mountain slope following the course set by the flags. You control your skier using the left and right arrow keys. The computer will ask you for the level of difficulty you wish to play. The number you input determines the initial distance between the flags, so the higher the number you input, the easier the level of play. We suggest you first try inputting 13 and as you become more confident, try playing with lower values. If you are a real thrill-seeker then see if you can complete the course with a level of difficulty of 7.

If you run into ten flags then you will be disqualified by the judges (ie the computer). Beware of *rocks* along the course.

Variables

a	Used to determine distance between flags
a$	Graphic for flag
c$	Graphic for skier
diff	Used to increase difficulty along course
x	Initial position for skier
j	Used to locate the flags
dist	Distance of course
f	Position for flag
m$	Used to check key pressed
spd	Pause
cr	Number of flags hit

Program

Line 10 prevents key repeat if a key is held down.

Line 90 generates a rock after a random interval of time.

Line 100 makes a continuous sound while skiing.

Lines 230–250 generate a number to move flags left or right.

Lines 330–340 test whether you have hit a flag or a rock. To do this we had to use TEST, so the screen positions

referenced are in High Resolution Mode.

Lines 360–380 make the course progressively narrower.

Lines 390–410 scan the keyboard and move the skier left or right.

Line 510 turns off the AFTER and EVERY commands used.

Line 600 increments the number of flags hit and checks whether you have hit ten.

Line 700 generates a rock at a random position.

Line 710 resets the random interval before another rock.

Explanation

This program introduces two BASIC procedures which are specific to the Amstrad.

The first one is the use of the internal clock by using the BASIC commands AFTER and EVERY. The Amstrad has an inbuilt real-time clock which is running all the time the computer is switched on. The AFTER and EVERY commands make it possible for you to send control of a program to a specified subroutine AFTER a given time has elapsed or EVERY so often. After the specified subroutine has been performed, control returns to the program where it left off. This feature is referred to as 'multi-tasking' because the computer appears to be performing more than one task at once. Your User Manual contains further information in the chapter on Interrupt Features (ch 10, p 1).

The second feature of Amstrad BASIC demonstrated in this program is using the command TEST to check a position on the screen. The screen display of the Amstrad is actually composed of two screen displays superimposed on each other. These are the Text Screen and the Graphics Screen. There are three different Text Screens which can be selected on the Amstrad using the MODE command.

MODE 0 specifies a Text screen of 20 characters across by 25 lines down
MODE 1 specifies a Text screen of 40 characters across by 25 lines down
MODE 2 specifies a Text screen of 80 characters across by 25 lines down

Irrespective of which mode the Text Screen is in, superimposed over it will always be the Graphics Screen which should be regarded as 640 pixels across the screen by 200 pixels down the screen. The term pixel refers to the smallest point addressable on the screen.

The resolution of the Graphics Screen is given as the number of pixels addressable across the screen by the number of pixels down the screen, so the resolution of the Graphics Screen is expressed as 640 by 200. However, this resolution is only achieved when the Text Screen is in MODE 2 (ie 80 character mode). When the Text Screen is in MODE 1 the resolution is only half this, 320 by 200, and in MODE Ø, the resolution is only 160 by 200. The awkward part of this is that in the lower resolution modes the addressing of individual pixels in the Graphics Screen is still done as though you were in the 640 by 200 resolution mode. This means that if for example you are in MODE Ø (20 characters across the Text Screen), the resolution of the Graphics Screen would be 160 by 200, but each pixel across the screen would not be numbered Ø to 159, but Ø to 636 in steps of 4, ie

MODE Ø Ø,4,8,12,16 ... 628,632,636 (160 NUMBERS)
MODE 1 Ø,2,4,6,8 ... 634,636,638 (320 NUMBERS)
MODE 2 Ø,1,2,3, ... 637,638,639 (640 NUMBERS)

This is why we said earlier that it is better to think of the Graphics Screen as always being 640 by 200, since you must always refer to screen positions in the Graphics Screen relative to this resolution, irrespective of the Text Screen.

A major difference between the Text and Graphics Screens is the direction in which they are referenced. The Text Screen is always referenced relative to the point 1, 1 located in the *top left* corner of the screen, whereas the Graphics Screen is referenced to the graphic position Ø, Ø situated in the *bottom left* corner of the screen.

The command TEST is used to determine the value of the ink colour at a specified pixel position in the Graphics Screen. This command has been used in this program to check whether the skiing character has collided with a rock or a flag. In this program the flags, the rocks and the skiing character are all being displayed in the Text Screen, but since the Text and

Graphics Screens are being displayed simultaneously on the screen, what is being displayed in the Text Screen can also be addressed as being within the Graphics Screen. This means that we can check the ink colour at any position in the Graphics Screen, even though it may have been displayed to the screen as a part of the Text Screen (ie by a PRINT statement).

```
9 REM skirun
10 SPEED KEY 255,255
20 SYMBOL AFTER 229
30 SYMBOL 248,56,56,146,254,146,170,170,
40
40 SYMBOL 229,48,60,62,60,48,32,32,32
50 INK 1,6:INK 2,26:INK 3,0:PAPER 2:BORD
ER 26
60 MODE 1
70 CLS
80 INPUT "level of play ";a
90 AFTER INT(RND(300)+200),2 GOSUB 700
100 GOSUB 660:EVERY 600,0 GOSUB 660
110 a$=CHR$(229)
120 c$=CHR$(248)
130 diff=0
140 x=20
199 REM GAME
200 j=20-INT(a/2)
210 FOR dist=1 TO 300
220 LOCATE 1,1:PRINT 300-dist
230 f=INT(RND(1)*40-a):IF f<=1 THEN GOTO
 230
240 IF f>15 THEN j=j+1
250 IF f<15 THEN j=j-1
260 IF j>40-a THEN j=j-1
270 IF j<1 THEN j=j+1
280 LOCATE j,25:PRINT a$:LOCATE j+a,25:P
RINT a$
290 PRINT
300 diff=diff+1
310 LOCATE X-1,7:PRINT "    "
320 LOCATE x-1,8:PRINT "    "
330 IF TEST (x*16-8,249)=1 THEN GOTO 600
340 IF TEST(x*16-8,249)=3 THEN GOTO 800
```

```
350 LOCATE x,9:PEN 3:PRINT c$:PEN 1
360 IF diff=100 THEN d=2:a=a-1
370 IF diff=175 THEN d=1:a=a-1
380 IF diff=250 THEN d=0:a=a-1
390 m$=INKEY$
400 IF m$=CHR$(242) THEN x=x-1
410 IF m$=CHR$(243) THEN x=x+1
420 FOR SPD =1 TO a:NEXT SPD
430 NEXT dist
440 FOR en=1 TO 10 :FOR pau=1 TO 500:NEX
T pau:LOCATE j,25:PRINT "FINISH":PRINT:N
EXT en
499 REM END OF GAME
500 CLS:PRINT "your score was ";diff/a-c
r*2
510 LET DD=REMAIN(0):FOR SND=150 TO 100
STEP -1:SOUND 129,SND,50,15:FOR P=1 TO 5
0:NEXT P:NEXT SND
520 SOUND 1,SND,100,15
530 BORDER 26
540 INPUT "would you like to play again
?";a$
550 IF LEFT$(A$,1)="Y" OR LEFT$(A$,1)="y
" THEN RUN
560 END
599 REM FLAG HIT ROUTINE
600 LET cr=cr+1 :IF cr=10 THEN GOTO 900
610 BORDER 8,12:AFTER 50,1 GOSUB 640
620 SOUND 2,1000,30,15
630 GOTO 350
640 BORDER 26:RETURN
650 REM SKI SOUND
660 SOUND 129,0,6000,10,0,0,15
670 RETURN
699 REM ROCK GENERATOR
700 LOCATE j+a/(RND(3)+1),25:PEN 3 :PRIN
T CHR$(244):PEN 1
710 AFTER INT(RND(300)+200),2 GOSUB 700
720 RETURN
799 REM ROCK HIT
800 BORDER 15,8:INK 4,15,8:PAPER 4
810 CLS:LOCATE 8,16:PRINT "YOU HAVE HIT
A ROCK ..."
820 LET DD=REMAIN(0)
```

```
830 FOR SND=1 TO 150 STEP 2:SOUND 129,SN
D,50,15:NEXT SND
840 SOUND 129,0,200,15,0,0,4
850 GOTO 910
899 REM DISQUALIFICATION
900 LOCATE 1,25:PRINT "YOU HAVE BEEN DIS
QUALIFIED FOR HITTING  TEN FLAGS. THE JU
DGES SUGGEST YOU TRY ANEASIER LEVEL."
910 PRINT:PRINT "      press return to c
ontinue"
920 BORDER 15,8
930 LET DD=REMAIN(0)
940 FOR g=1 TO 400:NEXT g
950 INPUT a
960 RUN
```

5 Emergency landing

The *USS Amsterprise* is running low on fuel reserves and must make an immediate stop on a nearby planet to refuel. You are the captain and it is up to you to land your ship, using what little fuel is available to you. The ship computer will help you by displaying your altitude, your downward velocity and the amount of fuel remaining. The computer will then ask you how much thrust you require from your retro-rockets. It will only allow you to input values for thrust in the range of 0–50000 to prevent the rockets from overheating. Through your port-hole you can see the rapidly approaching landing base.

There are five levels of difficulty and if you can land your ship at Level Five then you should put in an application to NASA!

Variables

diff	Level of difficulty. Used to determine the gravitational effect and fuel available
vel	Velocity
alt	Altitude
fuel	Fuel
used	Amount of fuel used for each thrust
d	Diameter of port-hole display
r	Radius of port-hole display
thr	Thrust
n	Counter of degrees for port-hole display

Program

Line 20 sets all angle references to degrees.

Lines 40–90 define windows for sections of screen display.

Line 170 sets all variables to initial values.

Line 230 checks if fuel has run out.

Line 240 checks if you have successfully landed.

Line 250 checks if you have crashed.

Line 290 calculates amount of fuel used.

Line 310 updates velocity.

Lines 400–470 draw port-hole display.

The display of the landing base is done by plotting points in concentric circles. These circles are only drawn if the altitude has dropped by 100 metres or more. Line 400 checks that the altitude has changed by 100 from when the display was previously drawn. Line 410 updates the value for d. Line 420 updates the value for the radius (r) so that as you approach the ground the landing base gets bigger.

Explanation

This program makes extensive use of the windowing function on the Amstrad to make displaying the information easier and the program easier to follow. The WINDOW command enables you to address a particular part of the screen to print to. This can be a lot easier than using the LOCATE command if you are always updating the same part of the screen. You can have up to eight windows, each one labelled with a number ranging from 0 to 7. When you first switch on your computer it defaults to Window 0, which would be equivalent to the command:

```
WINDOW #0,1,40,1,25
```

which is the entire screen. In the command, 0 after # is the window number, ie the label, followed by the leftmost position, the rightmost position, the highest position and the lowest position on the screen.

For example, if you put in the command:

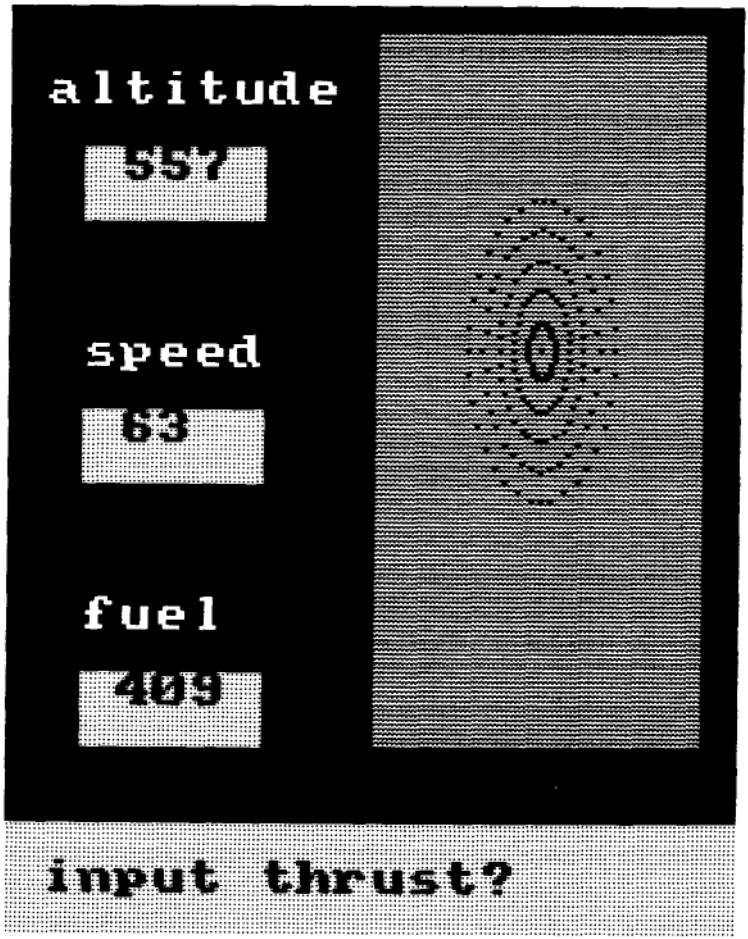
```
WINDOW #5,20,20,12,12
```

then you would have defined a window of only one character space in the centre of the screen.

In this program we have defined:

WINDOW #1 as your altitude display
WINDOW #2 as your velocity display
WINDOW #3 as your fuel display

WINDOW #4 as your port-hole
WINDOW #5 computer input line




```
9 REM EMERGENCY LANDING
10 MODE 0
20 DEG
30 INK 0,0:INK 1,26:INK 2,24:INK 3,23:IN
K 4,6,8:INK 5,22
40 WINDOW #0,1,20,1,25:PAPER #0,0:PEN #0
,1:CLS #0
50 WINDOW #1,3,7,5,6:PAPER #1,2:PEN #1,0
:CLS #1
60 WINDOW #2,3,7,12,13:PAPER #2,2:PEN #2
,0:CLS #2
```

```
70 WINDOW #3,3,7,19,20:PAPER #3,2:PEN #3
,0:CLS #3
80 WINDOW #4,11,19,2,20:PAPER #4,3:PEN #
4,4:CLS #4
90 WINDOW #5,1,20,23,25:PAPER #5,2:PEN #
5,0:CLS #5
100 LOCATE #0,2,3:PRINT "altitude"
110 LOCATE #0,3,10:PRINT "speed"
120 LOCATE #0,3,17:PRINT "fuel"
150 CLS #5:INPUT #5,"level of difficulty
 (1-5) ";diff:CLS #5
160 IF diff<1 OR diff>5 THEN 150
170 vel=100:alt=1000:fuel=400+100*diff:u
sed=0:d=0:r=0
199 REM GAME
200 LOCATE #1,1,1:PRINT #1,INT(alt)
210 LOCATE #1,1,1:PRINT #1,INT(alt)
220 LOCATE #2,1,1:PRINT #2,INT(vel)
230 LOCATE #3,1,1:PRINT #3,INT(fuel)
240 IF fuel<=0 THEN GOTO 520
250 IF INT(alt)<=0 AND vel<5 THEN 540
260 IF alt<=0 THEN 500
270 GOSUB 400
280 LOCATE #5,2,2:INPUT #5,"input thrust
";thr:CLS #5
290 IF thr<0 OR thr >50000 THEN 280
300 used=thr/50000*43
310 fuel=fuel-used
320 vel=vel+(vel*(2000+fuel)/50000+(diff
*2))-(thr/(2000+fuel))
330 alt=alt-vel
340 GOTO 200
399 REM LANDING BASE
400 IF INT(alt/100)=d THEN RETURN
410 d=INT(alt/100)
420 r=(10-d)*13
430 ORIGIN 464,248
440 FOR n=0 TO 360 STEP 10
450 PLOT r*SIN(n),r*COS(n),0
460 NEXT n
470 RETURN
499 REM MESSAGES
500 CLS #4:LOCATE #4,4,11:PRINT #4,"YOU"
:LOCATE #4,2,13:PRINT #4,"CRASHED"
```

```
510 GOTO 550
520 CLS #4:LOCATE #4,4,10:PRINT #4,"YOU"
:LOCATE #4,2,12:PRINT #4,"CRASHED":LOCAT
E #4,1,15:PRINT #4,"your fuel":LOCATE #4
,2,17:PRINT #4,"ran out"
530 GOTO 550
540 LOCATE #4,1,19:PRINT #4,"WELL DONE"
550 CLS #5:LOCATE #5,4,2:PRINT #5,"press
 any key"
560 IF INKEY$="" THEN 560
570 CLS #4:GOTO 150
```

6 Draw straws

Pit your wits against the computer. This is a computerised version of the game in which you have to draw straws with the aim of leaving another player with the last remaining straw. The catch with this one is that you are playing against the computer.

The computer will generate a random number of straws and then randomly decide who will go first. You and the computer will then take it in turns to draw straws. You can take either one, two or three straws at a time with the aim of leaving the computer with the last straw.

Variables

n	Number of straws
tk	Number of straws taken
you	Your score
me	Computer score

Program

Lines 10–50 define windows and ink colours. (Note that we have used the default ink values for ink 0, 1, 2 and 3.)

Line 60 generates a random number of straws.

Lines 80–100 generate graphics for straws.

Line 150 decides who goes first.

Lines 300–360 determine the computer's choice.

Lines 700–740 display number of straws remaining.

The graphics for the straws are generated in line 60. These graphics are stored in the string s$. Lines 710–730 print the string s$ from the first straw to the last remaining one (ie the *n*th one). The straw is printed three times on different lines to make it larger.

Explanation

This is an example of 'artificial intelligence'. Since a computer cannot actually think, a program in which a computer has to

make a decision, as in this one, is referred to as an Artificial Intelligence program. The decisions are based entirely upon decision lines within the program (lines 300 to 360). In this case the decisions are made using IF...THEN... statements. This technique becomes very unwieldy when the computer has to make a lot of decisions or decisions based on several different precepts.

To make this program more efficient in its decision making, you can make the following alterations:

Delete lines 300, 340, 350, 360, then type these lines in:

```
300 IF (n+3)/4=INT((n+3)/4) THEN tk = INT(RND(3))+1
310 IF (n+2)/4=INT((n+2)/4) THEN tk=1
320 IF (n+1)/4=INT((n+1)/4) THEN tk=2
330 IF n/4=INT(n/4) THEN tk=3
```

This technique is far more efficient and will work out the exact number of straws to take for virtually any number of straws. However, the decision to be made in this program is based upon only one precept — the number of straws remaining. More complicated artificial intelligence programs usually have to base their decisions upon several precepts, as in chess where the computer must base its decision on such information as whether it can take a piece with any of its men, whether a move will lose it a man and if it will, whether it would be worth it in the long run. In fact, the syntax checker inside your computer is an example of artificial intelligence where the decision is whether or not to accept a command.

Decision making on a larger scale is usually done using computer logic. On the Amstrad you can use the logical expressions (AND, OR, XOR, NOT). In addition to these logical expressions you can also make use of the comparative algebraic expressions (=, <, >, <>, <=, >=). All these expressions apply not only to numbers but also to strings of characters.

```
AND...  IF x=1 AND b=2 THEN...do something
```

The computer will *only* execute the command after the THEN statement if x = 1 *and* b = 2 (ie if both conditions are *true*).

```
OR...  IF x=1 OR b=2 THEN...do something
```

The computer will execute the command after the THEN statement if *either* x = 1 *or* b = 2 (ie if either condition is *true*).

XOR... IF x = 1 XOR b = 2 THEN... do something

The computer will *only* execute the command after the THEN statement if *one* condition is *true* and *one* condition is *false*.

NOT... IF NOT x = 1 THEN... do something

The computer will *only* execute the command after the THEN statement if x<>1 (ie the condition is *false*).

On the Amstrad, when it determines whether a condition is true or false, it gives it a value: 0 for false and −1 for true. This is referred to as *inverse logic*, since on most computers a true statement returns a value for the truth test of 1. The IF... THEN... statement can therefore be looked at as IF (−1) THEN..., where the operation after the THEN statement is only performed if the truth test (after the IF statement) returns a value of −1 (ie it is true). The logical expressions (AND, OR, XOR, NOT) alter the value of the truth test. Although we found that the Amstrad definitely uses inverse logic, the section in the User Manual on logical expressions (ch 4, p 18) describes it as *normal logic* (ie a true statement returns a 1).

You can always make use of truth testing in your own programming; in fact we have made extensive use of it in programs later in the book.

```
x=x+(x=1)
```

amounts to the same as:

```
IF x=1 THEN LET x=x-1
```

taking this one step further,

```
x=x+(x=1)-(x>2)-(x<0)
```

amounts to saying all three of the following lines:

```
IF x=1 THEN LET x=x-1
IF x>2 THEN LET x=x+1
IF x<0 THEN LET x=x+1
```

The use of inverse logic on the Amstrad appears to be an oversight in the programming of the BASIC ROM, since all other computers use the standard form of a truth test returning a value of +1 for true statements. So bear in mind that all the truth tests used in programs in this book are opposite to truth tests used on other computers.

```
9 REM DRAW STRAWS
10 CLS:INK 4,15
20 WINDOW #1,1,40,3,5:PEN #1,2
30 WINDOW #2,10,30,7,7
40 WINDOW #3,1,40,10,24
50 WINDOW #4,17,24,25,25:PEN #4,3
60 n=INT(RND(1)*15)+5
70 s$=""
80 FOR a=1 TO 20
90 s$=s$+CHR$(149)+" "
100 NEXT a
110 GOSUB 700
150 IF RND(1)<0.5 THEN GOSUB 600:GOTO 28
0
160 LOCATE #2,1,1:PRINT #2,"You can go f
irst"
199 REM YOUR CHOICE
200 LOCATE #3,5,1:PRINT #3,"There are ";
n;" straws remaining"
210  LOCATE #3,8,5:INPUT #3,"how many do
 you take";tk
220 CLS #2
230 CLS #3
240 IF tk <1 OR tk >3 THEN 210
250 n=n-tk
260 GOSUB 700
270 IF n=1 THEN 500
280  LOCATE #3,5,1:PRINT #3,"There are "
;n;" straws remaining"
299 REM COMPUTER'S CHOICE
300 tk=3
310 IF n=2 THEN tk=1
320 IF n=3 THEN tk=2
330 IF n=4 THEN tk=3
340 IF n=5 THEN tk=1
350 IF n=6 THEN tk=1
```

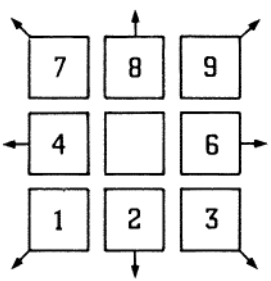
```
360 IF n=7 THEN tk=2
370 FOR a=1 TO 1000:NEXT
380 CLS #3
390 CLS #2
400 LOCATE #3,14,5:PRINT #3,"I will take
 ";tk
410 n=n-tk
420 GOSUB 700
430 IF n=1 THEN 530
440 FOR a= 1 TO 1500:NEXT
450 GOTO 200
499 REM SCORE UPDATE
500 LOCATE #4,1,1: PRINT #4,"YOU WIN"
510 you=you+1:GOSUB 800
520 GOTO 550
530 LOCATE #4,2,1: PRINT #4,"I WIN"
540 me=me+1:GOSUB 800
550 FOR a=1 TO 3000:NEXT
560 CLS #4
570 GOTO 60
599 REM MESSAGE
600 LOCATE #2,1,1:PRINT #2,"I will go fi
rst":RETURN
699 REM DISPLAY STRAWS
700 CLS #1
710 LOCATE #1,1,1:PRINT #1,LEFT$(s$,n*2)

720 LOCATE #1,1,2:PRINT #1,LEFT$(s$,n*2)
730 LOCATE #1,1,3:PRINT #1,LEFT$(s$,n*2)
740 RETURN
799 REM DISPLAY SCORES
800 LOCATE 1,1:PRINT "My score  ";me:LOC
ATE 20,1:PRINT "Your score  ";you
810 RETURN
```

7 Maze plays

Lost in a crossword puzzle. How quickly can you escape, and how many bonus chalices can you collect on your way?

There are five levels of difficulty in this game and once you have escaped you have the option not only of playing a new maze or the same one again but also of having the computer show an instant replay of all the moves you made. Controls for this game are on the numeric key-pad, giving diagonal movement as well as left, right, up and down.



You will be faced with the decision in this game between trying to complete the maze in the shortest possible time and collecting the bonus chalices. We have left it up to you to decide how you wish to approach this problem. Remember that the higher the level of difficulty, the harder the maze and the higher the value of the bonus chalices.

Variables

p(40,22)	Positions within the maze
z$(1000)	Stores the moves played for the replay option
df	Level of difficulty
diff	Maximum number of blocked positions per line
b	Random position across a line for inserting blocks
m	Random column for placing chalices
n	Random row for placing chalices
o	Column number of the position of your man
q	Row number of the position of your man

d	Number of moves made
bp	Number of chalices collected
t	Time at start of maze
tm	Total time taken in seconds

Program

Lines 110–160 draw the grid for the maze.

Lines 200–260 randomly generate positions for blocks and print them.

Lines 300–360 randomly generate positions for the chalices and print them.

Lines 370–390 print the finishing position and store it in the array.

Line 410 checks that the starting position is not blocked. If it is, then the program goes to a subroutine in line 580.

Line 430 prints your man.

Line 440 stores your position for the replay.

Line 470 checks whether you have pressed 'q'. This is an escape routine in case you have a maze where you are totally blocked in.

Line 480 increments the number of moves made.

Lines 500–530 update the values for the row and column position of your man. This is done using truth testing (see explanation below).

Line 540 prevents you from moving to a position where there is a block.

Line 550 checks whether you have collected a chalice.

Line 560 checks whether you have reached the end of the maze.

Line 580 is the subroutine which repositions your man at the start of the maze if the starting position is blocked, by moving you one space to the right.

Lines 590–620 reset the values of your position to those of the previous position if you have tried to move into a blocked square.

Line 700 determines the time taken to complete the maze.

Lines 760–790 display options.

Lines 800–850 replay the moves you have made (see explanation below).

Lines 900–950 replace the chalices for another game in the same maze.

Explanation

The array p(40,22) holds all the information for the maze, which consists of 40 columns of 22 rows. Each position within the maze therefore has its own position within the array. For a block in the maze we put a 1 in that position in the array. For example, if we have a block at column 3, row 4 then the value held in the array at p(3,4) would equal 1.

This is also done for the positions of the chalices, where these positions are given the value 2. For example, if we have a chalice at column 4, row 6 then the value held in array at p(4,6) would equal 2.

The same is done for the finishing position of the maze where the value held in the array is 3.

Lines 500 to 530 use a very complex form of logic to update the position of the man quickly and efficiently. Line 500 could have been written as a series of IF...THEN... statements, such as:

```
IF k$ = "7" then o = o - 1
IF k$ = "4" then o = o - 1
IF k$ = "1" then o = o - 1
```

and so on

From the previous chapter in this book you will know that the truth test on the Amstrad returns a value of -1 if the test is true and 0 if it is false. In line 500 then, if you have pressed 7 the truth test (k$ = "7") will return the value of -1, while the other truth tests in that line will all return the value of 0. Line 500 then could be read as:

```
500  o = o + (-1) + (0) + (0) - (0) - (0) - (0)
```

Pressing 7 is a diagonal control which has to update both the row and column positions, so '7' is used in the truth testing to update the values of both o and q.

Line 510 checks to make sure that your position is not outside the maze by using a further truth test using the comparative arithmetic operators. If the truth test is positive (ie -1), then the value of o will automatically be adjusted to keep you inside the maze.

Lines 520 and 530 have the same function as lines 500 and 510 but this time they update the value for the row position.

The array z$(d) holds the row and column numbers of the position you are in. Because d is the number of moves you have made, z$ actually holds each position that you have been in as you have moved through the maze. This is done by taking the values of the row and column positions (o and q) and converting them into a string stored in the array z$.

For example if your position in the maze is column 40, row 33 then the string held by z$ for that move number (d) would be:

chr$(o) + chr$(q)

which, in the computer's memory, would look like:

"(" + "!" = "(!"

The values from the array z$ are used in the game for the replay option and also to reset the values for o and q if you have made an invalid move, by reversing the process used to store them in z$:

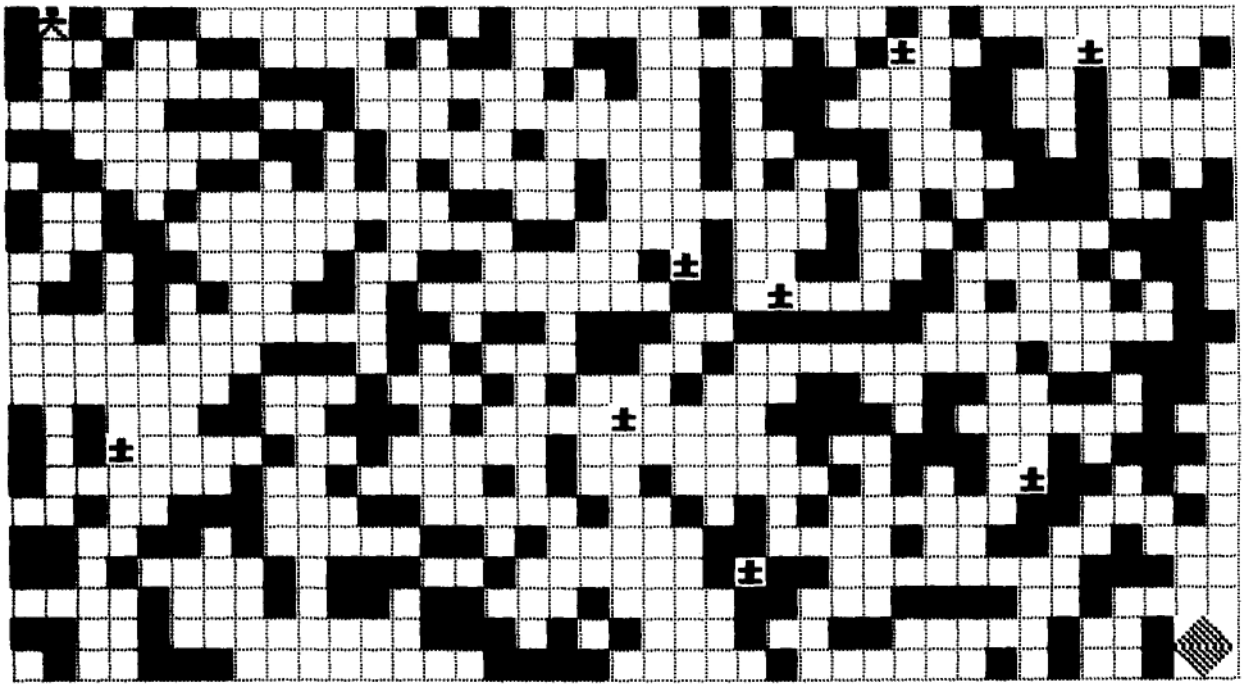
o = ASC(LEFT$(z$(d),1))
q = ASC(RIGHT$(z$(d),1))

We don't like to give high scores, but it is possible to complete the maze in under 10 seconds. Good luck!

```
9 REM MAZE PLAYS
10 CLS
20 ENV 1,10,1,1
30 DIM p(40,22)
40 DIM z$(1000)
50 INK 3,6
60 INPUT "difficulty level (1 to 5) ";df
:diff=df+15
99 REM DRAW GRID
100 CLS
110 FOR x=0 TO 624 STEP 16
120 PLOT x,48:DRAWR 0,352,2
```

```
130 NEXT
140 FOR y=47 TO 400 STEP 16
150 PLOT 0,y:DRAWR 624,0,2
160 NEXT
199 REM GENERATE MAZE
200 FOR a= 1 TO 22
210 FOR l= 1 TO diff
220 b=INT(RND(1)*39)+1
230 p(b,a)=1
240 LOCATE b,a:PRINT CHR$(143)
250 NEXT l
260 NEXT a
299 REM GENERATE CHALICES
300 FOR x=1 TO 8
310 m=INT(RND(1)*39)+1
320 n=INT(RND(1)*21)+1
330 IF p(m,n)=1 THEN 310
340 LOCATE m,n:PRINT CHR$(171)
350 p(m,n)=2
360 NEXT x
370 PEN 3:LOCATE 38,21:PRINT CHR$(222);C
HR$(223)
380 LOCATE 38,22:PRINT CHR$(221);CHR$(22
0):PEN 1
390 p(38,21)=3:p(39,21)=3:p(38,22)=3:p(3
9,22)=3
399 REM MAIN PROGRAM
400 o=1:q=1:d=1:bp=0
410 IF p(o,q)=1 THEN GOTO 580
420 t=TIME
430 LOCATE o,q:PRINT CHR$(249)
440 z$(d)=CHR$(o)+CHR$(q)
450 k$=INKEY$
460 IF k$="" THEN 450
470 IF k$="q" OR k$="Q" THEN 700
480 d=d+1
490 LOCATE o,q:PRINT CHR$(32)
499 REM UPDATE POSITION
500 o=o+(k$="7")+(k$="4")+(k$="1")-(k$="
9")-(k$="6")-(k$="3")
510 o=o-(o<1)+(o>39)
520 q=q+(k$="7")+(k$="8")+(k$="9")-(k$="
1")-(k$="2")-(k$="3")
```

```
530 q=q-(q<1)+(q>22)
540 IF p(o,q)=1 THEN GOTO 590
550 IF p(o,q)=2 THEN bp=bp+1:SOUND 1,200
,10,15,1
560 IF p(o,q)=3 THEN 700
570 GOTO 430
580 o=o+1:GOTO 410
590 d=d-1
600 o=ASC(LEFT$(z$(d),1))
610 q=ASC(RIGHT$(z$(d),1))
620 GOTO 430
699 REM PRINT SCORE & MENU
700 tm=(TIME-t)/300
710 FOR z=200 TO 100 STEP -4
720 SOUND 1,z,10,15
730 NEXT z
740 PEN 1
750 LOCATE 1,24:PRINT"You took ";INT(tm)
;"secs. and scored ";INT(3000-((tm+d)*20
)+bp*(50*(diff-15)))
760 INPUT " press   <r>  <n>  or  <p>  "
;s$
770 IF s$="r" THEN 800
780 IF s$="p" THEN 900
790 RUN
799 REM RETRACE
800 FOR i=1 TO d-1
810 o=ASC(LEFT$(z$(i),1))
820 q=ASC(RIGHT$(z$(i),1))
830 LOCATE o,q:PEN 3:PRINT CHR$(143)
840 FOR k=1 TO 300:NEXT k
850 NEXT i
860 GOTO 700
899 REM PLAY AGAIN
900 FOR x=1 TO 39
910 FOR y=1 TO 22
920 LOCATE x,y
930 IF p(x,y)=2 THEN PRINT CHR$(171)
940 NEXT y:NEXT x
950 GOTO 400
```



You took 21 secs. and scored 2550
press <r> <n> or <p> ?

8 Stop the invasion

How long can you hold back the invasion force from planet X? You are the sole surviving member of the Lunar Advance Warning Earth Defence task force. All the other members of your platoon are dead or dying and it is up to you to slow the invaders down as much as possible to give Earth enough time to prepare its own defences.

You move your photon cannon left and right using the 1 and Ø keys, and pressing the space bar releases a high-energy photon bolt. The longer you survive, the more difficult the game becomes. At the end of the game you will be given a score and your hit/miss ratio. If you can survive until the sixth attack wave and achieve a hit/miss ratio greater than 60 per cent then Earth will be safe.

Variables

a(1Ø)	Number of aliens in each attack wave
a$(1Ø)	Holds the information for each attack wave
sou	The tone period for music
b$	Your photon cannon
level	Number of the attack wave you are currently fighting
gpos	Position of your photon cannon
height	Height of the approaching aliens
ti	Number of moves aliens make to the left before they move closer
ke$	Key pressed
m	Random number for generating aliens on each level
fi	Holds the pixel position of your photon cannon
sh	Shots made in each attack wave
hh	Height of the photon torpedo
gh	Used to determine positions on the screen for displaying the torpedo.
hits	Number of aliens hit in each attack wave

hitsm	Total number of aliens hit
shm	Total number of shots made
pr	Duration of notes for music at end of game

Program

Lines 20–30 dimension arrays to hold number of aliens in each wave (a) and the string containing the aliens (a$).

Lines 50–60 define the volume envelopes used in the music and shooting routines.

Line 70 sets the keyboard scan so that no key repeat will occur. This is so that you cannot hold a key down for repetitive firing or moving.

Lines 80–100 define graphic characters for aliens and your cannon.

Lines 170–200 read notes from DATA statement and play music.

Lines 220–240 draw base force shield.

Line 320 moves the aliens (see explanation below).

Line 350 scans the keyboard. If a key has been pressed then the program goes to line 700.

Line 380 advances the aliens one line.

Line 390 checks whether aliens have reached your force shield.

Lines 500–600 generate the number of aliens and their relative position in all ten attack waves (see explanation).

Line 710 moves your cannon one space to the right.

Line 720 moves your cannon one space to the left.

Line 810 calculates the position of your cannon in High Resolution Mode.

Line 820 cannon shot sound.

Lines 840–870 move and display photon torpedo.

Lines 880–890 check whether the torpedo hit an alien.

Line 910 keeps aliens moving while torpedo is on screen.

Line 1000 deletes alien that was hit from a$ by replacing it with a space.

Line 1010 sound for explosion.

Line 1030 decrements the number of aliens in the wave. If there are none left, then advance to next level.

Lines 1720–1730 determine final score and percentage of hits made.

Explanation

This program makes use of a sideways scrolling routine to move the aliens. The scrolling routine can be used to move any string of characters from right to left across the screen. It is a one line routine:

```
9000 CLS
9010 a$="HELLO THIS IS THE SIDEWAYS
SCROLL.    "
9020 a$=MID$(a$,2,LEN(a$))+LEFT$(a$,1)
9030 LOCATE 1,1: PRINT a$
9040 GOTO 9020
```

If you then type GOTO 9000 to start this routine, the message stored in a$ will scroll from right to left on your screen. This is done by removing the first character in a$, shifting the other characters one position to the left within the string a$ and adding the first character to the end of the string. That is, a$ above after line 9020 will become:

"ELLO THIS IS THE SIDEWAYS SCROLL. H"

and after running through this line a few times, a$ will become:

"THE SIDEWAYS SCROLL. HELLO THIS IS"

The command MID$(a$,2,LEN(a$)) takes the string a$ from the second position onwards. The command LEFT$(a$,1) takes the first character in the string a$. So:

```
a$ = LEFT$(a$,1) + MID$(a$,2,LEN(a$))
```

will make no change at all to the string a$. However, by

reversing this, the first character moves to the end of the string.

A sideways scroll to the right is done with virtually the same command, ie

a$ = RIGHT$(a$,LEN(a$)) + MID$(a$,1,LEN(a$) – 1)

Play around with this routine using different messages. Try changing line 9030 to:

LOCATE 1,1: PRINT LEFT$(a$,10)

The routine from line 500 to line 600 generates the positions for the aliens within the string a$. There are two loops in this routine—c, which is the number of the wave currently being generated and x, which is the position of the alien or blank space being placed within the string for that wave. Line 530 generates a random number between 0 and 3. If this number is less than 1 then an alien is placed in that position within the string, otherwise a space is placed there. Line 550 inserts the space in the string. Line 590 inserts an alien in the string and increments the variable (a) holding the number of aliens in the string for that wave.

```
9 REM STOP THE INVASION
10 CLS
20 DIM a(10)
30 DIM a$(10)
40 INK 2,26:INK 3,1:INK 1,24:INK 0,0:BOR
DER 0:PAPER 0
50 ENV 1,10,-1,10
60 ENV 2,30,-1,15
70 SPEED KEY 255,255
80 SYMBOL AFTER 217
90 SYMBOL 217,16,56,124,238,238,186,186,
254
100 SYMBOL 218,129,66,60,66,90,36,36,24

110 PRINT "ONE MOMENT PLEASE...  INITIAL
IZING."
120 LOCATE 11,11:PRINT "USE"
130 LOCATE 15,12:PRINT "1    for LEFT"
140 LOCATE 15,14:PRINT "0    for RIGHT"
150 LOCATE 13,16:PRINT "SPACE  for FIRE"
160 GOSUB 500
```

```
170 RESTORE :FOR zz=1 TO 2:FOR zx=1 TO 1
6
180 READ sou:SOUND 2,sou,50,15,1
190 NEXT zx
200 RESTORE :NEXT zz
210 CLS
220 b$=CHR$(217)
230 PEN 2:FOR p1=1 TO 40 STEP 2:LOCATE p
1,18:PRINT CHR$(222);CHR$(223):NEXT p1
240 FOR p1=1 TO 40 STEP 2:LOCATE p1,19:P
RINT CHR$(220);CHR$(221):NEXT p1:PEN 1
250 level=1
260 gpos=20
270 height=3
299 REM START OF GAME
300 FOR ti=0 TO 60
310 LOCATE 1,height:PRINT a$(level)
320 a$(level)=MID$(a$(level),2,40)+LEFT$
(a$(level),1)
330 SOUND 129,1000,5,10
340 LOCATE gpos,25:PRINT b$
350 ke$=INKEY$:IF ke$<>"" THEN 700
360 NEXT ti
370 LOCATE 1,height:PRINT SPACE$(40)
380 height=height+1
390 IF height=18 THEN GOTO 1700
400 GOTO 300
500 REM initialize
510 FOR c=1 TO 10
520 a$(c)="":a(c)=0
530 FOR x=1 TO 40
540 m=INT(RND(1)*3)
550 IF m<1 THEN GOTO 600
560 a$(c)=a$(c)+" "
570 NEXT x
580 NEXT c
590 RETURN
600 a$(c)=a$(c)+CHR$(218):a(c)=a(c)+1
610 GOTO 570
699 REM CHECK FOR KEYS
700 LOCATE gpos,25:PRINT " "
710 IF ke$="0" THEN gpos=gpos-(gpos>2)
720 IF ke$="1" THEN gpos=gpos+(gpos<39)
730 IF ke$=" " THEN 800
```

```
740 GOTO 340
799 REM SHOOT ROUTINE
800 LOCATE gpos,25:PRINT b$
810 fi=gpos*16-8
820 SOUND 129,100,20,10,1,0,8
830 sh=sh+1:LOCATE 1,1:PEN 2:PRINT "SHOT
S- ";sh:PEN 1
840 FOR gh=1 TO 4
850 FOR hh=(gh-1)*100 TO gh*100 STEP 8


860 IF gh=1 THEN hh=hh+16
870 PLOT fi,hh,2:PLOT fi,hh,0
880 IF TEST(fi+2,hh)=1 THEN 1000
890 IF TEST(fi-2,hh)=1 THEN 1000
900 NEXT hh
910 a$(level)=MID$(a$(level),2,40)+LEFT$
(a$(level),1):LOCATE 1,height:PRINT a$(l
evel)
920 NEXT gh
930 GOTO 340
999 REM HIT ROUTINE
1000 LET a$(level)=LEFT$(a$(level),(gpos
-1))+" "+RIGHT$(a$(level),(40-gpos))
1010 SOUND 129,200,60,15,2,0,15
1020 hits=hits+1:LOCATE 20,1:PEN 2:PRINT
 "HITS-";hits:PEN 1
1030 a(level)=a(level)-1:IF a(level)=0 T
HEN GOTO 1100
1040 GOTO 340
1099 REM END OF LEVEL
1100 CLS:RESTORE :LOCATE 18,10:PRINT "we
ll done"
1110 PRINT:PRINT "you have completed lev
el ";level
1120 PRINT :PRINT "YOU MADE ";sh; "SHOTS
 AND HIT ";hits
1130 hitsm=hitsm+hits:shm=shm+sh:sh=0:hi
ts=0
1140 PRINT:PRINT "here we go again...."
1150 FOR zz=1 TO 2
1160 FOR zx=1 TO 16
1170 READ sou:SOUND 2,sou,50,15,1
1180 NEXT zx
```

```
1190 RESTORE:NEXT zz
1200 CLS:level=level+1
1210 PEN 2:FOR p1=1 TO 40 STEP 2:LOCATE
p1,18:PRINT CHR$(222);CHR$(223):NEXT p1
1220 FOR p1=1 TO 40 STEP 2:LOCATE p1,19:
PRINT CHR$(220);CHR$(221):NEXT p1:PEN 1
1230 GOTO 300
1499 REM DATA FOR SOUND
1500 DATA 60,40,60,40,47,50,53,56
1510 DATA 60,40,60,40,45,45,30,30
1520 DATA 478,100,478,100,478,50,478,100
,402,100,426,50,426,100,478,50,478,100,5
06,50,478,200
1699 REM END OF GAME
1700 CLS :LOCATE 5,10:PRINT "YOU HAVE BE
EN DESTROYED.."
1710 shm=shm+sh:hitsm=hitsm+hits
1720 PRINT:PRINT "FINAL SCORE = ";hitsm*
level:PRINT
1730 PRINT "HIT/MISS RATIO IS ";INT(hits
m/shm*100);"%"
1740 RESTORE 1520
1750 FOR zz=1 TO 11
1760 READ sou:READ pr
1770 SOUND 1,sou,pr,15,2
1780 NEXT zz
1790 PRINT:PRINT "PRESS ANY KEY FOR ANOT
HER GO...":
1800 IF INKEY$="" THEN GOTO 1800
1810 RUN
```

9 Triathlon

You are competing for Amstradia in the triathlon event at the Home Computer Olympics. The triathlon consists of running races in the 100 metres, 200 metres and 400 metres. You run by pressing any two keys alternately on your keyboard. To select the event you wish to run first, simply press the number key corresponding to the event.

It is *impossible* to break out of this program using the ⟨ESC⟩ key. To break the program, you must first complete the race then by pressing *q* you can escape.

Bear in mind that if you stop running with your fingers then your competitor will stop running on the screen but the clock will keep going. So keep on pounding that keyboard with those fingers.

By the way, the world record for the 100 metres event (set in March 1985) is 10.16 seconds.

Variables

d(4)	Array holding high scores
body$(3)	Array holding the three characters used to display the three body positions
leg$(3)	Array holding the three characters used to display the three leg positions
a$	Running Track
x	Loop counter for music and change of ink colour
dd$	Keyboard scan for menu selection
so	The tone period for music
tm	Metres to run
z	Counter for metres run
t	Initial time
c$	Keyboard scan for running
d$	Holds value of previous key pressed
x	Time at end of loop
y	Cumulative time taken
r$	Keyboard scan for return to menu at end of race
cou	Counter for body, leg and track movement

Program

Lines 50–290 define graphics characters for the running person and the track and redefine the number characters to look like a digital display.

Symbols:				
	Head	220		221
	body1	222		223
	body2	232		233
	body3	232		233
	leg1	230		231
	leg2	226		227
	leg3	228		229
	big face	218		219
	numbers 0 to 9	48	to	57

Lines 310–330 generate the running track.

Lines 580–630—demonstration mode.

Lines 700–710 establish distance to be run from menu selection. Since the formula at line 710 uses the value of the key pressed, the value for key 3 must be set to 4 in line 700.

Lines 720–760 delete all menu information.

Line 770 prints the stadium roof and then prints a back space (CHR$(8)), so that the next print command begins at the start of the next line.

Lines 780–800 print the crowd.

Line 860—see explanation.

Line 870 sets a random pause between GET SET and GO.

Line 1000 reprints the head.

Line 1010—the sound of the starter's gun.

Line 1050—see explanation.

Lines 1060–1080 scan the keyboard for keys pressed to run with. Line 1080 stores the value of the key pressed, c$ in b$. Then the next time through the loop, line 1070 checks that

the same key has not been pressed. This is so that you must run with alternating keys.

Line 1090—sound of footsteps.

Line 1130 determines total time taken by incrementing y by the difference in time between each key press.

Line 1140 displays time taken to two decimal places, using the PRINT USING command.

Line 1200 prints big happy face.

Line 1210—sound of applause.

Line 1220 displays final time.

Line 1230 checks whether time taken is less than the current high score and updates the high score if it is.

Lines 1240–1260—keyboard scan to return to main menu. This is the *only* place where you can break out of the program by pressing the *q* key.

Line 1300 alternates the characters used for printing the body and the legs.

Line 1330—sideways scroll routine for the track.

Line 1400—sound for false start.

Explanation

CALL &BB00. This is a direct ROM CALL. The basic keyword CALL is used to link machine code programs to BASIC programs. The CALL command can access machine code programs written by the user, or access the inbuilt machine code routines written in the Amstrad's ROM. The number following the CALL command is the address in memory where the routine is situated. The '&' in front of the number signifies to the computer that the number is in HEXadecimal notation (base 16) and not in DECimal notation (base 10) which is normally used for numbers.

This particular call accesses a routine in the upper ROM of the Amstrad, which completely clears the keyboard buffer. You will have found by now that sometimes within a program you can press keys faster than the program can respond. This

is because each key press is placed on a queue, to be read by the program when it next requests a key input. This routine therefore empties the queue each time it is called. A major feature of the routine is that it also ignores the ⟨ESC⟩ key, so if you wish to make use of this routine in your own programs make sure that you have some way of breaking out of the program (see line 125Ø).

There are many routines such as this built into the Amstrad's ROM. The addresses for these routines can be found in the Amstrad Firmware Manual.

This program makes extensive use of the SYMBOL command not only for the graphic displays, but also to redefine the whole character set for the numbers. It is even possible using the SYMBOL command to redefine the entire character set if, for example, you wanted to write in Arabic letters. This is a lot of hard work, but on the Amstrad using the SYMBOL command it is far easier than on most other popular home computers.

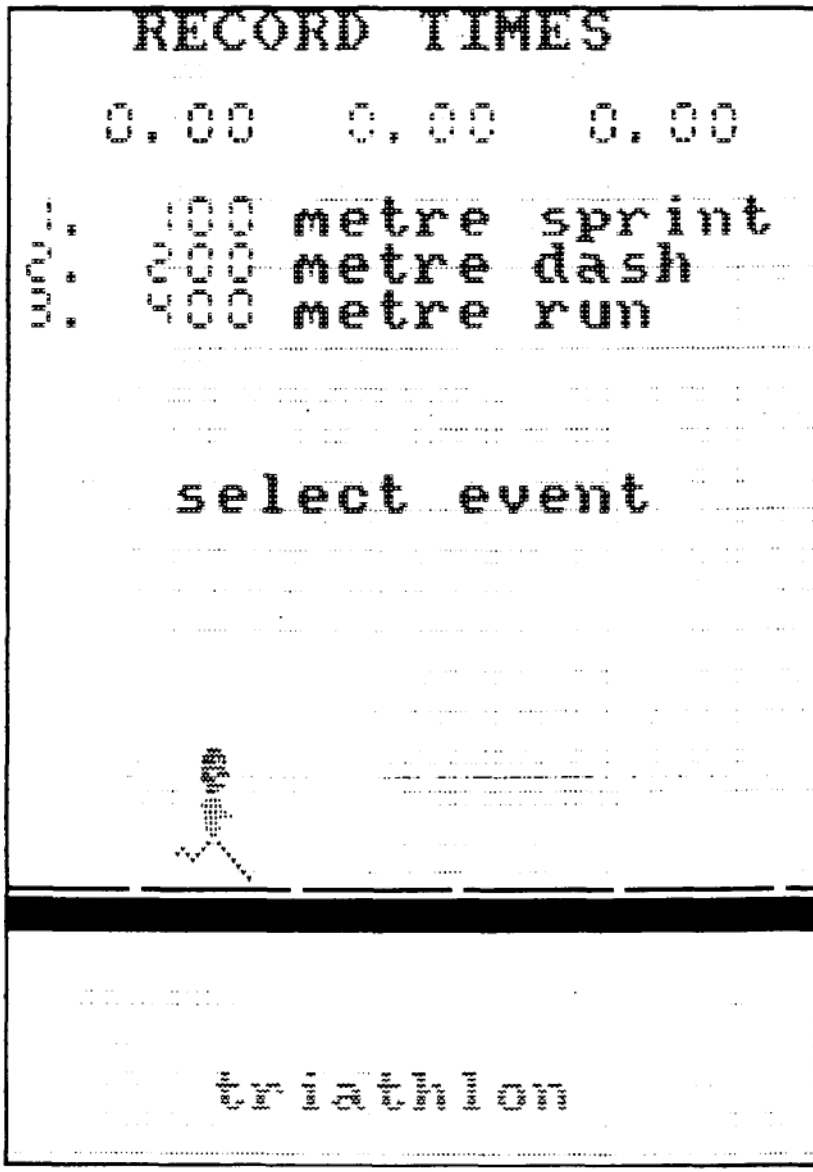
```
9 REM TRIATHLON
1Ø CLS:MODE Ø
2Ø DIM d(4)
3Ø INK 2,16:INK 3,18:INK 4,14:INK 5,15:I
NK 7,2
4Ø DIM body$(3):DIM leg$(3)
5Ø SYMBOL AFTER 48
6Ø SYMBOL 222,3,5,9,17,9,5,1,1 :SYMBOL 2
34,192,255,Ø,Ø,Ø,Ø,Ø,Ø
7Ø SYMBOL 223,192,226,212,2ØØ,192,192,19
2,192:SYMBOL 235,Ø,255,Ø,Ø,Ø,Ø,Ø,Ø
8Ø SYMBOL 218,7,15,12,15,13,14,15,7
9Ø SYMBOL 219,24Ø,248,152,248,216,56,248
,24Ø
1ØØ SYMBOL 226,1,2,4,8,16,32,16,Ø
11Ø SYMBOL 227,64,48,12,4,8,16,56,Ø
12Ø SYMBOL 228,1,34,84,8,Ø,Ø,Ø,Ø
13Ø SYMBOL 229,64,32,16,8,4,2,1,Ø
14Ø SYMBOL 22Ø,1,3,3,3,3,3,1,1
15Ø SYMBOL 221,192,224,16Ø,224,112,128,2
24,128
16Ø SYMBOL 23Ø,Ø,Ø,Ø,Ø,Ø,1,1,Ø
17Ø SYMBOL 231,64,96,8Ø,96,192,64,64,96
```

```
180 SYMBOL 232,1,3,3,5,3,1,1
190 SYMBOL 233,192,192,224,248,192,192,1
92,192
200 SYMBOL 48,14,17,17,0,17,17,14,0
210 SYMBOL 49,0,1,1,0,1,1,0,0
220 SYMBOL 50,14,1,1,14,16,16,14,0
230 SYMBOL 51,14,1,1,14,1,1,14,0
240 SYMBOL 52,0,17,17,14,1,1,0,0
250 SYMBOL 53,14,16,16,14,1,1,14,0
260 SYMBOL 54,14,16,16,14,17,17,14,0
270 SYMBOL 55,14,1,1,0,1,1,0
280 SYMBOL 56,14,17,17,14,17,17,14,0
290 SYMBOL 57,14,17,17,14,1,1,14,0
300 a$=""
310 FOR x=1 TO 5
320 a$=a$+CHR$(234)+CHR$(235)+CHR$(235)+
CHR$(235)
330 NEXT x
340 body$(1)=CHR$(222)+CHR$(223)
350 FOR x=2 TO 3
360 body$(x)=CHR$(232)+CHR$(233)
370 NEXT x
380 leg$(3)=CHR$(228)+CHR$(229)
390 leg$(2)=CHR$(226)+CHR$(227)
400 leg$(1)=CHR$(230)+CHR$(231)
499 REM MENU
500 LOCATE 5,17:PEN 2:PRINT CHR$(220);CH
R$(221):PEN 1
510 LOCATE 4,1:PRINT "RECORD TIMES"
520 LOCATE 1,3:PRINT USING "###.##";d(1)
;d(2);d(4)
530 LOCATE 1,5:PRINT "1. 100 metre sprin
t"
540 LOCATE 1,6:PRINT "2. 200 metre dash"
550 LOCATE 1,7:PRINT "3. 400 metre run"
560 LOCATE 5,11:PRINT "select event "
570 RESTORE
580 FOR x=1 TO 36
590 dd$=INKEY$:IF dd$>"0" AND dd$<"4" TH
EN 700
600 LOCATE 6,24:INK 8,INT(x/2)+4:PEN 8:P
RINT "triathlon"
610 READ so:SOUND 1,so,30,15:SOUND 2,956
,5,13,0,0,2:SOUND 4,so+1,30,15
```

```
620 GOSUB 1300
630 NEXT x:RESTORE:GOTO 580
699 REM PRINT STADIUM
700 IF dd$="3" THEN dd$="4"
710 tm=VAL(dd$)*33
720 LOCATE 4,1:PRINT SPACE$(13)
730 LOCATE 1,5:PRINT SPACE$(20)
740 LOCATE 1,6:PRINT SPACE$(20)
750 LOCATE 1,7:PRINT SPACE$(20)
760 LOCATE 6,24:PRINT SPACE$(9)
770 LOCATE 1,10:PEN 7:PRINT "/\/\/\/\/\/
\/\/\/\/\";CHR$(8):PEN 2
780 FOR x=1 TO 60
790 PRINT CHR$(224);
800 NEXT x
810 PAPER 3:PRINT "                    "
:PAPER 0
820 PEN 1
830 LOCATE 1,3:PRINT"  ON YOUR MARKS   "

840 FOR pause=1 TO 1000:NEXT
850 LOCATE 3,3:PRINT"   GET SET    "
860 CALL &BB00
870 FOR pause=1 TO INT(RND(5)*500+500):I
F INKEY$<>"" THEN GOTO 1400
880 NEXT
890 LOCATE 3,3:PRINT"      GO       "
999 REM MAIN PROGRAM
1000 LOCATE 5,17:PEN 2:PRINT CHR$(220);C
HR$(221)
1010 SOUND 1,1000,15,15,0,0,12
1020 FOR z=1 TO tm
1030 t=TIME
1040 GOSUB 1300
1050 CALL &BB00
1060 c$=INKEY$:IF c$="" THEN 1060
1070 IF b$=c$ THEN 1060
1080 b$=c$
1090 SOUND 129,1000,2,15
1100 GOSUB 1300
1110 x= TIME
1120 GOSUB 1300
1130 y=y+x-t
```

```
1140 LOCATE 1,1:PEN 1:PAPER 5:PRINT USIN
G "##.##";y/300;:PRINT " ":PAPER 0
1150 NEXT z
1199 REM TIME
1200 LOCATE 5,17:PEN 2:PRINT CHR$(218);C
HR$(219):PEN 1
1210 SOUND 1,0,300,15,0,0,15
1220 LOCATE 1,22:PRINT"YOUR TIME WAS ";:
PRINT USING "##.##";y/300
1230 IF y/300<D(VAL(dd$)) OR d(VAL(dd$))
=0 THEN d(VAL(dd$))=y/300
1240 LOCATE 5,24:PRINT"PRESS ENTER":LET
r$=INKEY$
1250 IF r$="q" THEN SPEED KEY 30,2:END
1260 IF r$<>CHR$(13) THEN 1240
1270 y=0:CLS :GOTO 500
1299 REM RUNNING ROUTINE
1300 cou=cou+1:IF cou=4 THEN cou=1
1310 LOCATE 5,18:PEN 3:PRINT body$(cou)
1320 LOCATE 5,19:PEN 2:PRINT leg$(cou)
1330  a$=MID$(a$,2,20)+LEFT$(a$,1)
1340 PEN 5:PAPER 4:PRINT a$:PAPER 0
1350 RETURN
1399 REM FALSE START
1400 SOUND 1,600,60,15
1410 CLS:LOCATE 5,10:PRINT"FALSE START"
1420 FOR PAUSE=1 TO 1000:NEXT
1430 CLS:GOTO 770
1499 REM DATA FOR MUSIC
1500 DATA 478,358,319,284,319,319,319,37
9,379,379,0,0,478,358,319,284,319,319,31
9,379,379,379,0,0,379,358,379,426,478,47
8,478,478,478,478,478,478
```


RECORD TIMES
0.00 0.00 0.00
100 metre sprint
200 metre dash
400 metre run
select event
triathlon

10 Sound envelope generator

This program is designed to help you learn how to use the sound commands on the Amstrad. Making your computer produce sound is one of the hardest aspects of programming, but once you have mastered it you will find it one of the most fascinating. The sound chip which the Amstrad uses is in our opinion one of the best on the market and is similar to the one that is used in the BBC micro. Although at first it appears that it is a rather complicated procedure to produce a sound, by making full use of the Volume and Tone envelope commands you have far greater control of the sound you are creating. The ENV command defines the Volume envelope and the ENT command defines the Tone envelope. The Volume envelope (ENV) is used to create the Attack, Decay, Sustain and Release (ADSR) curve used in simulating sounds. The Tone envelope (ENT) is used to create the rising and falling of the pitch within the sound. For example to make a laser-type sound you would want the pitch of the note to fall rapidly to give it a Doppler effect. Doppler is the name given to the effect that movement has on the pitch of sound. The typical example given for the Doppler effect is that of a steam train, where as the train approaches the pitch of the note rises, then as the train moves away the pitch drops. (See example below.)

This program helps you to define Volume and Tone envelopes and then try them. The envelopes are also displayed graphically.

Using the program

From the Master menu you would initially select option 1 to create a Volume envelope. You will then be asked which 'Envelope no.' you wish to define. This is because you can define up to five different Volume envelopes within the program. You will then be asked for the 'No. of sections' you wish to define in your envelope. Each section consists of a 'Step count', being the number of steps, a 'Step size', being the increase or decrease in volume for each step, and a 'Pause time', being the time taken for each step. This takes the same format as is used in the Sound Primer in your manual (ch 6, p 8). The legal values for each of these parameters are also shown in your manual.

After defining a Volume envelope you will be automatically returned to the Master menu. From here you would now select either option 1 to create another Volume envelope or redefine a previous one, or option 2 to create a Tone envelope. You are then prompted through the creation of the Tone envelope in exactly the same way as with the Volume envelope. A point to bear in mind when defining envelopes is that the time taken for the Tone Envelope should be the same time taken for the Volume envelope if you want the entire envelope to be played.

Option 3 on the Master menu takes you to the part of the program where you can try out the envelopes you have defined. You will first be asked which envelope numbers you wish to hear. The graphs of the envelopes will then be displayed, along with their definitions. A note will then be played using these envelopes. Control of the program is then handed back to you with a new menu offering:

r	to repeat the note
m	to return to the Master menu
p	to go on to the next section of the program where you can try different notes using any of the keys
c	to select a different combination of Volume and Tone envelopes (assuming you have defined several)
n	to add an amount of noise to your sound

Simply press the key corresponding to your selection.

If you select option p, the screen will clear and you will now be in Keyboard Mode. Your computer keyboard will now function as a musical instrument where each key pressed will play a different note using the envelopes you have defined and selected. Pressing the space bar will return you to the previous screen. Some keys will, however, break the program with a TYPE MISMATCH error, in which case simply type in:

```
GOTO 1200
```

to return to the Master menu without losing the envelopes you have already defined.

Examples

Try these sample envelopes:

```
ENV 1,1,0,40                          (1 section)
ENT 1,40,1,1,                         (1 section)

ENV 1,1,0,40                          (1 section)
ENT 2,3,-127,1,40,10,1                (2 section)

ENV 1,1,0,40                          (1 section)
ENT 3,10,-1,1,10,1,1,20,-1,1          (3 section)

ENV 2,5,20,2,5,-10,1,5,0,2,5,10,1,5,-20,2  (5 section)
ENT 4,1,0,40                          (1 section)
```

Try various combinations of these envelopes. Remember when entering them that after inputting the envelope number you will have to tell the computer how many sections you will be defining.

Variables

a(5,5)	Step count for Volume envelopes
b(5,5)	Step size for Volume envelopes
c(5,5)	Pause time for Volume envelopes
d(5,5)	Step count for Tone envelopes
e(5,5)	Step size for Tone envelopes
f(5,5)	Pause time for Tone envelopes
n(5)	Number of sections within each Volume envelope
o(5)	Number of sections within each Tone envelope
ve	Volume envelope number
vt	Time taken for Volume envelope (used to determine duration of note)
te	Tone envelope number
tt	Time taken for Tone envelope

noi	Noise level
i$	Reads menu selection
in$	Reads keyboard for characters
in	Stores character as its ASCII code
x	Input for selection from Master menu

Program

Line 40 dimensions arrays for holding the values for the Volume envelopes. Each of the arrays (a, b, and c), holds the value for the section in each envelope. For example a(1,1) holds the step count for the first section of ENV 1, b(1,1) holds the step size for the first section of ENV 1 and c(1,1) holds the pause time for the first section of ENV 1. Similarly, c(5,5) would hold the value for the pause time for the fifth section of ENV 5.

Line 50 dimensions arrays for holding the values for the Tone envelopes. The arrays d, e and f, correspond to the arrays a, b and c, for the Volume envelopes.

Line 60 holds the number of sections used in each envelope.

Line 70 sends you to the Master menu. It is better programming where possible to put your main menu at the end of the program and your subroutines at the beginning.

Lines 100–140 draw the axes for the Volume envelope graph.

Line 180 resets the values for the selected envelope number.

Line 230 sends you to the section that draws the envelope on the graph.

Lines 250–280 determine the duration of the Volume envelope.

Line 300 defines ENV to the sound processor.

Lines 340–350 draw the envelope on the graph.

Lines 400–460 draw the axes for the tone Envelope graph.

Line 500 resets the values for the selected envelope number.

Line 550 sends you to the section that draws the envelope on the graph.

Lines 570–610 determine the duration of the Tone envelope.

Line 620 defines ENT to the sound processor.

Lines 660–670 draw the envelope on the graph.

Lines 730–760 print out the Volume and Tone envelopes you have selected to play.

Lines 770–840 display the selected Volume and Tone envelope graphs.

Line 850 sets the noise level to zero.

Line 860 plays a note using the selected ENV and ENT.

Lines 870–950—music-playing menu.

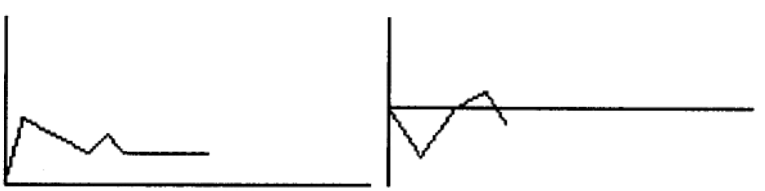
Lines 960–980—routine for adding noise.

Lines 1110–1140 read the keyboard for keys pressed, store key pressed as a string (in$), check whether the space bar has been pressed. Convert in$ to its ASCII value (in).

Line 1150 plays sound using the selected ENV and ENT with the tone of the note determined by 'in'.

Lines 1200–1230—Main menu.

Line 1240 sends you to the relevant subroutine.



```
ENV 1 , 10 , 20 , 5 , 10 , -10 , 20 , 1 ,
  50 , 50 , 1 , -50 , 50 , 1 , 0 , 250
ENT 1 , 10 , -15 , 10 , 10 , 10 , 15 , 10 ,
  10 , 5 , 10 , 10 , -10 , 5
```

```
r to repeat      m for menu      p to play
   c to change ENV/ENT      n for noise
```

```
9 REM SOUND ENVELOPE GENERATOR
10 MODE 1:BORDER 0
20 INK 0,0:INK 1,26
30 PEN 2
40 DIM a(5,5):DIM b(5,5):DIM c(5,5)
50 DIM d(5,5):DIM e(5,5):DIM f(5,5)
60 DIM n(5):DIM o(5)
70 GOTO 1200
99 REM VOLUME ENVELOPE
100 ORIGIN 32,384
110 DRAW 0,-256
120 DRAWR 555,0
130 LOCATE 1,1:PRINT"VOLUME"
140 LOCATE 36,18:PRINT"TIME"
150 WINDOW #1,1,40,20,25
160 CLS #1:INPUT #1,"Envelope no.",ve
170 CLS #1:INPUT #1,"No. of sections ",n
(ve)
180 FOR z=1 TO 5:a(z,ve)=0:b(z,ve)=0:c(z
,ve)=0:NEXT z
190 FOR il=1 TO n(ve)
200 CLS #1:INPUT #1,"Step count ";a(il,v
e)
210 CLS #1:INPUT #1,"Step size ";b(il,ve
)
220 CLS #1:INPUT #1,"Pause time ";c(il,v
e)
230 IF il=1 THEN GOSUB 340 ELSE GOSUB 35
0
240 NEXT il
250 vt=0
260 FOR tl=1 TO n(ve)
270 vt=vt+a(tl,ve)*c(tl,ve)
280 NEXT tl
290 CLS #1:PRINT #1,"Time used for env";
ve;"is ";vt
300 ENV ve,a(1,ve),b(1,ve),c(1,ve),a(2,v
e),b(2,ve),c(2,ve),a(3,ve),b(3,ve),c(3,v
e),a(4,ve),b(4,ve),c(4,ve),a(5,ve),b(5,v
e),c(5,ve)
310 LOCATE 2,25:PRINT"press any key"
320 WHILE INKEY$="":WEND
330 CLS:RETURN
```

```
340 ORIGIN 32,128
350 DRAWR a(il,ve)*c(il,ve),a(il,ve)*b(i
l,ve):RETURN
399 REM TONE ENVELOPE
400 ORIGIN 32,384
410 DRAW 0,-256
420 MOVER 0,128
430 DRAWR 555,0
440 LOCATE 1,1:PRINT"TONE"
450 LOCATE 1,2:PRINT"PERIOD"
460 LOCATE 36,9:PRINT"TIME"
470 WINDOW #1,1,40,20,25
480 CLS #1:INPUT #1,"Envelope no. ",te
490 CLS #1:INPUT #1,"No. of sections ",o
(te)
500 FOR z=1 TO 5:d(z,te)=0:e(z,te)=0:f(z
,te)=0:NEXT z
510 FOR il=1 TO o(te)
520 CLS #1:INPUT #1,"Step count ";d(il,t
e)
530 CLS #1:INPUT #1,"Step size ";e(il,te
)
540 CLS #1:INPUT #1,"Pause time ";f(il,t
e)
550 IF il=1 THEN GOSUB 660 ELSE GOSUB 67
0
560 NEXT il
570 tt=0
580 FOR tl=1 TO o(te)
590 tt=tt+d(tl,te)*f(tl,te)
600 NEXT tl
610 CLS #1:PRINT #1,"Time used for ent i
s ";tt
620 ENT te,d(1,te),e(1,te),f(1,te),d(2,t
e),e(2,te),f(2,te),d(3,te),e(3,te),f(3,t
e),d(4,te),e(4,te),f(4,te),d(5,te),e(5,t
e),f(5,te)
630 LOCATE 2,25:PRINT"press any key"
640 WHILE INKEY$="":WEND
650 CLS:RETURN
660 ORIGIN 32,256
670 DRAWR d(il,te)*f(il,te),d(il,te)*e(i
l,te):RETURN
```

```
699 REM PLAY SOUND
700 CLS:LOCATE 1,11:INPUT "Which Volume
Envelope";ve
710 CLS:LOCATE 1,11:INPUT "Which Tone En
velope";te
720 CLS
730 LOCATE 1,13:PRINT"ENV";ve;
740 FOR dl=1 TO n(ve):PRINT",";a(dl,ve);
",";b(dl,ve);",";c(dl,ve);:NEXT dl
750 LOCATE 1,16:PRINT"ENT";te;
760 FOR dl=1 TO o(te):PRINT",";d(dl,te);
",";e(dl,te);",";f(dl,te);:NEXT dl
770 ORIGIN 32,390:DRAW 0,-128:DRAWR 275,
0:ORIGIN 32,262
780 FOR il=1 TO n(ve)
790 DRAWR a(il,ve)/2*c(il,ve)/2,a(il,ve)
/2*b(il,ve)/2
800 NEXT il
810 ORIGIN 320,390:DRAW 0,-128:ORIGIN 32
0,322:DRAW 275,0:ORIGIN 320,322
820 FOR il=1 TO o(te)
830 DRAWR d(il,te)/2*f(il,te)/2,d(il,te)
/2*e(il,te)/2
840 NEXT il
850 ENV ve,a(1,ve),b(1,ve),c(1,ve),a(2,v
e),b(2,ve),c(2,ve),a(3,ve),b(3,ve),c(3,v
e),a(4,ve),b(4,ve),c(4,ve),a(5,ve),b(5,v
e),c(5,ve)
860 ENT te,d(1,te),e(1,te),f(1,te),d(2,t
e),e(2,te),f(2,te),d(3,te),e(3,te),f(3,t
e),d(4,te),e(4,te),f(4,te),d(5,te),e(5,t
e),f(5,te)
870 noi=0
880 SOUND 1,255,0,7,ve,te,noi
890 LOCATE 1,23:PRINT"r to repeat    m f
or menu    p to play"
900 LOCATE 1,25:PRINT"   c to change ENV
/ENT    n for noise  "
910 i$=INKEY$
920 IF i$="r"THEN GOTO 880
930 IF i$="m" THEN CLS:RETURN
940 IF i$="c"THEN GOTO 700
950 IF i$="n"THEN GOTO 980
```

```
960 IF i$="p"THEN GOSUB 1100:GOTO 720
970 GOTO 910
980 LOCATE 1,20:INPUT "LEVEL OF NOISE (0
-15)";noi
990 IF noi<0 OR noi>15 THEN GOTO 980
1000 GOTO 880
1099 REM PLAY MUSIC
1100 CLS:LOCATE 7,12:PEN 2:PRINT "PRESS
SPACE BAR TO RETURN"
1110 in$=INKEY$
1120 IF in$="" THEN GOTO 1110
1130 IF in$=CHR$(32) THEN RETURN
1140 in=ASC(in$)
1150 SOUND 129,in*20-900,0,7,ve,te,noi
1160 GOTO 1110
1199 REM MENU
1200 LOCATE 7,10:PRINT"1  set up volume
envelope"
1210 LOCATE 7,12:PRINT"2  set up tone en
velope"
1220 LOCATE 7,14:PRINT"3  play note"
1230 LOCATE 7,20:INPUT "input menu selec
tion";x:CLS
1240 ON x GOSUB 100,400,700
1250 GOTO 1200
```

11 Teledex

This program gives you a computerised telephone directory. You can display names, addresses and phone numbers on your computer screen in the same way as you have them in your address book. To find someone's address or phone number just press the key corresponding to their alphabetical page.

You can save files on tape or disk, load them back in at later times and add names, delete names or make alterations very simply with no complicated programming routines.

You move around the pages using the arrow keys. Using the ⟨SHIFT⟩ key with the right and left arrow keys jumps your cursor across to the next column. When you wish to enter some information, position the cursor where you want it and simply type in the information. To select a new page press the ⟨SHIFT⟩ and ⟨COPY⟩ keys together. The computer will then ask which page letter you want to go to. To delete any information or to make alterations all you do is position the cursor under the material to be deleted or changed and type over it, using the space bar to make deletions.

To save your file press ⟨CTRL⟩ and S together. This initiates the SAVE routine and will then prompt you through using the cassette player as usual.

To load a file there are two options. One is to load the file immediately on running the program. The other is to load the file during the program by pressing the ⟨CTRL⟩ and L keys together.

Important: To break out of the program, you must first go to the SAVE or LOAD routine and only then press ⟨BREAK⟩.

Variables

an$	Answer to load request
x	Variable used for loops to initialise memory
g	Variable used for loops to initialise memory
h	Variable used for loops to initialise memory
g	Counter used to display fields
c	Records per column
j	Fields per record
a$	Data for field headings

w	Window number
x	Horizontal cursor position
y	Vertical cursor position
p$	Page letter
addr	Address in memory for start of data
pr	Address in memory for start of selected page letter
pk	Window (column) currently in use
pp	Counter to display the information for the current page
px	Counter—same as 'pp'
hg	Counter—same as 'pp'
k$	Key pressed
page$	New page selected
data	File name used for saving and loading

Program

Line 10 sets the position of the highest point in memory available to the basic program. This makes the memory from 10000 onwards available for storing our information.

Line 50—UPPER$ converts the input to capital letters, the line then checks if the input was 'y', 'Y', 'yes', 'YES' actually anything that starts with 'y' or 'Y'.

Line 70—here because the program takes a while to initialise, so be patient.

Line 80 fills the first 100 locations used for information storage with spaces.

Lines 90–120—loop set up to fill memory with the record separating lines and spaces.

Lines 210–270 define the windows used on screen:

WINDOW #0 information line across top
WINDOW #1,3,5 labels for fields (name, address)
WINDOW #2,4,6 contain the information.

Lines 290—350 read and print the labels for each column (WINDOW #1,3,5,).

Lines 360–380 set the variables to the initial page.

Lines 390–440 take the information out of memory and display it on the screen for the selected page.

Line 450 prints the letters of the page currently being displayed.

Line 500 locates and prints the cursor.

Line 510 clears the keyboard buffer.

Line 540 checks for CTRL–S and CTRL–L.

Line 550 displays information at the cursor position. (Otherwise the cursor would be deleting the information on the screen.)

Lines 560–570 update the cursor position using logic.

Line 580 checks for valid key inputs.

Line 590 checks whether the ⟨ENTER⟩ key was pressed.

Line 600 updates window used.

Lines 610–660 ensure that the updated variables are valid.

Line 630 checks for the ⟨COPY⟩ key.

Line 680 displays the key pressed.

Line 690 stores the key pressed in memory as its ASCII value.

Line 700 sets k$ so that the cursor moves to the right.

Line 730 sets the start address for selected page.

Line 850—note that we have used 'data' as a label. Generally it is not possible to use a reserved word as a label but in this case 'data' is not used as a variable but as a string.

Explanation

This program is an example of a static data structure. In previous programs we have made use of arrays to store information. This program achieves the same result using memory locations as pigeon holes for storing information. Each hole has a number which is used to store and recall the contents. These memory locations will only hold numbers,

so to store a letter you must store it as a number (ie its ASCII value). For an explanation of ASCII code see your manual (ch 1, p 7). Values stored in memory locations can be in the range of 0 to 255.

Values are stored in memory locations using the POKE command and retrieved using the PEEK command. Not all the memory is available for storage in this manner as parts of it have already been set aside for the basic program, the screen display, the ROM flags, etc. The part of memory used for storage of information is always immediately above the basic program area. By using the MEMORY command you can set the top of the basic program area (RAM top) and thus increase the amount of memory available for storage. If you try to use the area of memory above HIMEM then your information will tend to be corrupted by the work space the computer uses for calculation, keyboard buffer, etc. In your own programs do not be afraid to experiment with memory locations, as the worst thing that can happen is that you'll have to switch off the computer and start again. You cannot do any damage to the computer itself.

The advantage of using memory locations for storage is that information can be stored and retrieved much faster and with more control over the selection than if you're using another storage technique such as arrays. Although at first it appears that addressing the information location required is complicated, it is actually quite simple because you can address the required memory location as a function of the cursor position on the screen, as in line 550:

```
550 LOCATE #w,x,y:PRINT #w,CHR$(PEEK
(addr+(x-1)+(y-1)*20)+((w/2)-1*480))
```

'LOCATE #w,x,y' positions the cursor. The variable 'addr' holds the value of the memory address which holds the first piece of information for the current page. So when you PRINT CHR$(PEEK(addr)) you are displaying the character at the first position of the current page. X and y are the horizontal and vertical positions of the cursor, y is multiplied by 20 because there are 20 characters in each field of each record. So from 'addr' to 'addr + 20' we have the information stored at the positions where y = 1 and the values for x from 1 to 20. W is the current window being used and each window holds 480 characters.

At the first position on the screen, x = 1, y = 1, w = 2, so at this point
addr + (1 − 1) + (1 − 1)*20 + ((2/2) − 1*480) will equal 'addr'.

At the last position on the screen x = 20, y = 23, w = 6, so at this point
addr + (20 − 1) + (23 − 1)*20 + ((6/2) − 1*480) will equal 'addr + 1419'.

In line 730 'addr' is given the value for the start address of the page selected:

```
730  addr = INT((ASC(p$) − 65)/2)*1440 + 10000
```

'p$' is the character of which you wish to display the page. 'ASC(p$)' returns the ASCII value of the character you have chosen. This value is then reduced by 65. The ASCII value of 'A' is 65 and since we want this operation to return a zero for the 'A' page we then subtract 65. The number 1440 is the number of locations per page and 10000 is the first location where the information is stored. In order to fit all the information required in memory, we have displayed two letters per page (eg A and B). So that line 730 will also work if you select the second of the two letters (eg B), after taking the ASCII value and subtracting 65 from it, we then divide the number by 2 and take the INTeger part of the value returned. This means that 'addr' will be the same if you have selected either of A or B, C or D, E or F, etc.

We have used several of the print control characters in this program. (See your manual, ch 9, p 2.) In line 500 'PRINTing CHR$(22) + CHR$(1)' sets Transparent Mode so that the cursor does not mask the character underneath. 'PRINTing CHR$(22) + CHR$(0)' turns the Transparent Mode off. We have also had to use 'PRINT CHR$(8)' several times to prevent the cursor from moving to the next line, which would affect the display.

Further uses

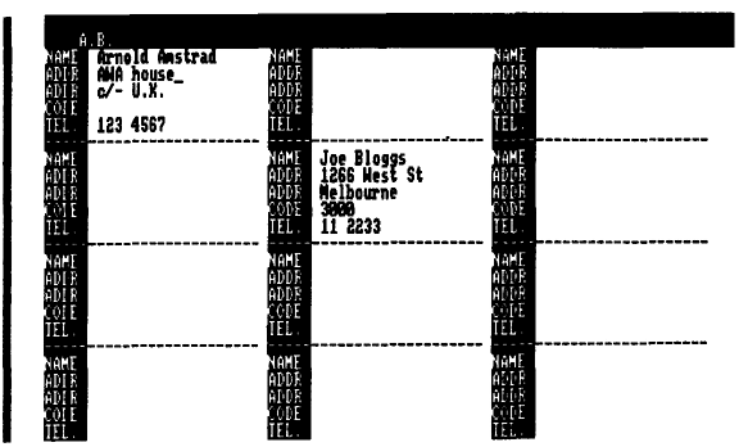
We have set this program up for use as a telephone directory, but there is absolutely no effort involved in altering the program to use it as a club record controller, an index of your video tapes or software programs or for any other filing purposes. To do this all you need to do is change the field

titles in the DATA statement in line 890. This is possible because the program is set up for free-form data entry by pages rather than by fields. When you make such alterations remember the labels can only be up to four characters long.

```
5 REM TELEDEX
9 REM INITIALISE MEMORY
10 MEMORY 9999
20 PAPER 0:PEN 1
30 CLS
40 INPUT "do you want to load a file";an
$
50 an$=UPPER$(an$):IF LEFT$(an$,1)="Y" T
HEN 850
60 MODE 0:LOCATE 8,12:INK 3,27,6:PEN 3
70 PRINT "wait"
80 FOR x=10000 TO 10100:POKE x,32:NEXT x
90 FOR x=10120 TO 28620 STEP 120
100 FOR g=0 TO 20:POKE x-g,45:NEXT g
110 FOR h=x TO x+100:POKE h,32:NEXT h
120 NEXT x
199 REM INITIALISE SCREEN
200 MODE 2
210 WINDOW #0,1,80,1,2:PAPER #0,1:PEN #0
,0:CLS #0
220 WINDOW #1,1,5,3,25:PAPER #1,1:PEN #1
,0:CLS #1
230 WINDOW #2,6,26,3,25:PAPER #2,0:PEN #
2,1
240 WINDOW #3,27,31,3,25:PAPER #3,1:PEN
#3,0:CLS #3
250 WINDOW #4,32,52,3,25:PAPER #4,0:PEN
#4,1
260 WINDOW #5,53,57,3,25:PAPER #5,1:PEN
#5,0:CLS #5
270 WINDOW #6,58,80,3,25:PAPER #6,0:PEN
#6,1
280 G=6
290 FOR c=1 TO 4
300 FOR j=1 TO G
310 READ a$
320 FOR w=1 TO 5 STEP 2
```

```
330 PRINT #w,a$;CHR$(8)
340 IF C=3 THEN LET G=5
350 NEXT w:NEXT j:RESTORE:NEXT c
360 x=1:y=1:w=2:p$="A":addr=10000
370 pr=addr
380 pk=2:CLS #pk
390 FOR pp=pr TO pr+1440 STEP 480
400 FOR px=0 TO 459 STEP 20:FOR HG=0 TO
19
410 PRINT #pk,CHR$(PEEK(pp+px+HG));
420 NEXT HG:PRINT #PK:NEXT px:pk=pk+2
430 IF pk>6 THEN GOTO 450
440 CLS #pk:NEXT pp
450 CLS #0:IF INT(ASC(p$)/2)=ASC(p$)/2 T
HEN PRINT #0,"    ";CHR$(ASC(p$)-1);".";p
$ ELSE PRINT #0,"    ";p$;".";CHR$(ASC(p$
)+1)
499 REM MAIN PROGRAM
500 LOCATE #w,x,y:PRINT #w,CHR$(22)+CHR$
(1);"_";CHR$(22)+CHR$(0)
510 CALL &BB00
520 k$=INKEY$
530 IF K$="" THEN GOTO 520
540 IF k$=CHR$(19) OR k$=CHR$(12) THEN 8
00
550 LOCATE #w,x,y:PRINT #w,CHR$(PEEK(add
r+(x-1)+(y-1)*20+((w/2)-1)*480))
560 x=x+(k$=CHR$(242))-(k$=CHR$(243))
570 y=y+(k$=CHR$(240))-(k$=CHR$(241))
580 IF k$>CHR$(31) AND k$<CHR$(123) THEN
 GOTO 680
590 IF k$=CHR$(13) THEN x=21
600 w=w+((k$=CHR$(246))*2)-((k$=CHR$(247
))*2)
610 w=w+((w>6)*6)-((w<2)*6)
620 y=y+(x<1)-(x>20)
630 IF k$=CHR$(224) THEN.GOTO 710
640 x=x+((x>20)*20)-((x<1)*20)
650 y=y+(y>23)-(y<1)
660 IF Y=23 AND X=20 THEN LET X=X-1
670 GOTO 500
680 LOCATE #w,x,y:PRINT #w,k$
690 POKE addr+(x-1)+(y-1)*20+((w/2)-1)*4
80,ASC(k$)
```

```
700 k$=CHR$(243):GOTO 560
710 CLS #0:INPUT "ENTER PAGE LETTER ";pa
ge$
720 p$=UPPER$(LEFT$(page$,1))
730 addr=INT((ASC(p$)-65)/2)*1440+10000
740 x=1:y=1:w=2
750 GOTO 370
799 REM LOAD, SAVE, DATA
800 CLS
810 IF k$=CHR$(19) THEN 870
820 INPUT "are you sure you want to LOAD
 !";an$
830 an$=UPPER$(an$)
840 IF LEFT$(an$,1)="N" THEN 450
850 LOAD "data"
860 GOTO 200
870 SAVE "data",b,10000,18620
880 GOTO 450
890 DATA "NAME","ADDR","ADDR","CODE","TE
L.","-----"
```



12 Australiana Smith and the forbidden temple

This program is a text adventure of the type that initiated the whole concept of using computers for entertainment. In this particular adventure you take the part of the hero in an African adventure. A missionary has been taken prisoner by the evil witch doctor and it is up to you to rescue her. You tell the computer what you wish to do—it is your eyes and your action, it tells you what you see and obeys your commands. You can move North, South, East, West, Up and Down and collect objects using Take. There will be many objects scattered throughout the adventure which you can take, some of which will help you, some of which are treasures and some of which are of no value at all. You will also encounter various hazards which you'll either have to Fight or Run from. When you Run, you will lose all the objects which you are carrying, and there is no guarantee that you will not be caught. If you elect to Fight, then the outcome of the fight depends on any weapons that you are carrying at the time and the strength of your opponent.

There are three keys scattered throughout the adventure, and you will need all three to release the missionary when you find her. Another feature of this adventure is that you can use the Look command to repeat the description of the present location.

Playing tips

When playing adventures it is advisable to draw a map as you go. If you are in one location and you go North, going South again will not always return you to your previous location and a map will provide a record of your moves.

Initially concentrate on collecting weapons as you explore the temple. Once you have found the missionary, go back and collect the keys. After releasing the missionary you should make your way to the exit, collecting treasure as you go.

You can only carry three objects at a time, so be selective.

Remember that your adventure begins outside the gate, so once you've entered the temple grounds going south from any of the locations level with the gate will take you out of the temple grounds and out of the adventure.

Variables

n	Counter used for location number
rmn	Number of locations
t	Number of hazards
b(42,3)	Holds the object number in each room b(x,1) holds general objects b(x,2) used for manipulation of objects b(x,3) holds the hazards
rm$(40)	Location descriptions
a$(42)	Object description
mv(42,6)	Holds the moves available from each location
ob(4)	Holds the objects you are carrying
in$	Used to display instructions
hz$	Holds description of hazards
r	Random number for locating objects
q$	Input from keyboard
vc$	String holding all the legitimate entries
x	Used to interpret keyboard entry into a command
flt	Flag indicating whether you have elected to Run
cn	Number of keys held
q	Input from keyboard for dropping an object
th	Number of the object that you've dropped
vl	Value of your score
rf	Your fighting strength
ch	A random chance

Program

Line 20 sets location number to begin at location 1.

Line 30 defines number of locations and number of hazards.

Line 40 dimensions arrays.

Lines 50–80 read and display instructions.

Lines 90–110 load room descriptions into the array rm$.

Lines 120–140 load object descriptions into the array a$.

Lines 150–170 load hazard descriptions into the array hz$.

Lines 180–200 load moves available from each location into the array mv.

Lines 210–220 select random location for missing missionary.

Lines 230–270 select random locations for objects.

Lines 280–320 generate random locations for hazards.

Line 420 displays location description and object at that location.

Lines 460–500 check that the input is valid and select routine to be executed.

Lines 510–530 check whether movement is allowable (99 is used to indicate that you cannot go there).

Line 540 sets location number according to the direction selected.

Lines 610–670 calculate number of keys you are carrying.

Lines 770–790 determine the number of objects you are carrying.

Lines 820–830 pick up the object and replace it with object 9 (object 9 is a dead rat).

Lines 950–970 replace object you wish to drop with object you wish to pick up.

Lines 1010–1050 calculate and print the value of treasures taken.

Lines 1080–1190 reset all variables for new game.

Lines 1310–1340 determine your fighting strength from the objects you are carrying.

Lines 1400–1410 determine chance of escaping if you elect to Run.

Lines 1500–1520 determine chance of winning if you elect to Fight.

Explanation

Adventure programs are probably the definitive examples of complex data structures where you have several different arrays all being accessed simultaneously, using a common reference, in this case n, the room number. In an adventure every location is given a number. When you elect to move to another location (NSEWUD) all the program needs to do is to specify the number of the new location.

By using separate arrays for all the items of information, it is possible to generate an adventure which will play differently each time you play it. We have done this so that the objects, the missionary and the hazards will not necessarily appear in the same place each time you play the game. However the location of each room and the directions available from any given location will always be the same.

There are two main arrays around which the entire game is centred, the array holding the description for each location (rm$) and the array holding the moves available from each location (mv). The array mv at a particular location number will look something like this:

99 32 14 99 22 99

This corresponds to the locations you would go to if you entered:

N S E W U D

So in this example you couldn't go North. South would take you to location 32, East would take you to location 14 and you couldn't go West. Up would take you to location 22 and you couldn't go Down.

The routine in lines 460 to 490 determines which position along the line within this array for this location you have selected and consequently you will be moved to the corresponding location number.

Alterations

If you find the adventure too difficult to complete, you may like to increase your chances. This can be done quite simply by altering lines 1400 and 1500. You might like to try:

```
1400 ch = INT(RND(1)*3 + 3)
1500 ch = INT((RND(1))*rf) + 3
```

You can create your own adventure, using exactly the same layout, by changing the room descriptions in the data statements from lines 1800 to 2190 and the object descriptions in lines 2300 to 2840. If you do this, the first three objects are required to release the missionary, objects number 4 to 30 are miscellaneous objects, objects 31 to 35 are the weapons objects, objects 36 to 39 are treasures, object 40 is the goal (in this case the missing missionary) and the last five objects are the hazards.

Note: The data statements from lines 2900 to 3050 hold either twelve or six numbers separated by commas. These lines hold the data for allowable movements (array mv). If when you run your program a particular location describes an exit to the east, for example, and you cannot go to the east, the most likely place you'll find the error is in these data lines.

```
5 REM ADVENTURE
9 REM INITIALISE
10 CLS
20 n=1
30 READ rmn,t
40 DIM b(42,3),rm$(40),a$(42),mv(42,6),o
b(4)
50 FOR i=1 TO 5
60 READ in$
70 PRINT in$:PRINT:PRINT
80 NEXT i
90 FOR i=1 TO rmn
100 READ rm$(i)
110 NEXT i
120 FOR i=1 TO rmn
130 READ a$(i)
140 NEXT i
150 FOR i=1 TO t
160 READ hz$(i)
170 NEXT i
180 FOR i=1 TO rmn
190 READ mv(i,1),mv(i,2),mv(i,3),mv(i,4)
,mv(i,5),mv(i,6)
200 NEXT i
210 r=INT(RND(1)*26+9)
```

```
220 b(r,1)=rmn
230 FOR i=1 TO (rmn-1)
240 r=INT(RND(1)*rmn+1)
250 IF b(r,1)<>0 THEN 240
260 b(r,1)=i
270 NEXT i
280 FOR i=1 TO t
290 r=INT(RND(1)*34+3)
300 IF b(r,3)<>0 THEN 290
310 b(r,3)=i
320 NEXT i
399 REM MAIN PROGRAM
400 INPUT "press enter to start";q$
410 FOR s=1 TO 25:PRINT:NEXT s
420 PRINT rm$(n);" ";a$(b(n,1))
430 PRINT
440 INPUT "what now ";q$
450 PRINT :PRINT
460 vc$="NSEWUDTL"
470 FOR zz=1 TO LEN(vc$)
480 IF q$=MID$(vc$,zz,1) THEN x=zz
490 NEXT zz
500 ON x+1 GOTO 1650,510,510,510,510,510
,510,730,410
510 IF mv(n,x)<>99 THEN 540
520 PRINT "you can't go that way ":PRINT
530 GOTO 430
540 n=mv(n,x)
550 flt=0
560 IF n=1 THEN 1000
570 PRINT rm$(n);" ";a$(b(n,1)):PRINT
580 flt=0
590 IF b(n,1)<>rmn THEN 700
600 PRINT "behind a tripled-locked door
":::
610 FOR i =1 TO 3
620 FOR j=1 TO 3
630 IF ob(i)<>j THEN 650
640 cn=cn+1
650 NEXT j
660 NEXT i
670 IF cn=3 THEN 1620 ELSE 750
680 cn=0
690 flt=0
```

```
700 IF b(n,3)=0 THEN 430
710 PRINT :PRINT:PRINT
720 GOTO 1300
730 IF flt=1 THEN 1600
740 IF flt<>1 THEN 770
750 PRINT "you only have ";cn;" keys ":
760 GOTO 680
770 FOR i=1 TO 3
780 IF ob(i)=0 THEN 820
790 NEXT i
800 PRINT "you can only carry three obje
cts -leave one behind "
810 GOTO 900
820 ob(i)=b(n,1)
830 b(n,1)=9
840 PRINT "it's yours"::
850 GOTO 430
899 REM DROP OBJECT
900 PRINT "1. ";a$(ob(1)):PRINT "2. ";a$
(ob(2)):PRINT "3. ";a$(ob(3)):PRINT "4.
";a$(b(n,1))::
910 INPUT "leave which number ";q
920 PRINT ::
930 IF q>4 THEN 910
940 IF q=4 THEN 430
950 th=ob(q)
960 ob(q)=b(n,1)
970 b(n,1)=th
980 GOTO 430
999 REM END OF GAME
1000 PRINT "leaving so soon ? value of i
tems taken  is ";
1010 FOR i=1 TO 3
1020 IF ob(i)<rmn-t+1 THEN 1040
1030 vl=vl+ob(i)*1000
1040 NEXT i
1050 PRINT vl:PRINT:PRINT
1060 INPUT "like to try again ";q$
1070 IF MID$(q$,1,1)="N" THEN 1210
1080 cn=0
1090 n=1
1100 CLS
1110 FOR i=1 TO rmn
1120 b(i,1)=0
```

```
1130 b(i,2)=0
1140 ob(1)=0
1150 ob(2)=0
1160 ob(3)=0
1170 v1=0
1180 b(i,3)=0
1190 NEXT i
1200 GOTO 210
1210 END
1299 REM RUN OR FIGHT
1300 PRINT hz$(b(n,3)):PRINT
1310 FOR h=1 TO 3
1320 IF (ob(h)<30)+(ob(h)>rmn-t) THEN 13
40
1330 rf=rf+ob(h)-30
1340 NEXT h
1350 INPUT "run or fight ";q$
1360 PRINT ::
1370 IF MID$(q$,1,1)="R" THEN 1400
1380 IF MID$(q$,1,1)="F" THEN 1500
1390 GOTO 1350
1400 ch=INT(RND(1)*3+1)
1410 IF ch<>1 THEN 1440
1420 PRINT "you were caught ! ":PRINT
1430 GOTO 1530
1440 PRINT "you've escaped but lost ever
ything you  were carrying.":PRINT
1450 ob(1)=0
1460 ob(2)=0
1470 ob(3)=0
1480 flt=1
1490 GOTO 430
1500 ch=INT((RND(1))*rf)+1
1510 rf=0
1520 IF ch>(b(n,3)*2) THEN 1550
1530 PRINT "you lost !":PRINT:PRINT
1540 GOTO 1060
1550 PRINT "you won !":PRINT:PRINT
1560 b(n,3)=0
1570 GOTO 430
1599 REM MESSAGES
1600 PRINT "no time for that. You're too
 busy       running."
1610 GOTO 430
```

```
1620 PRINT "you've rescued ";a$(rmn);" n
ow you must escape with any treasure you
 carry":PRINT:PRINT
1630 ob(1)=rmn
1640 GOTO 430
1650 PRINT "use N S E W U D T or L pleas
e"::
1660 GOTO 430
1699 REM SIZE & INSTRUCTIONS
1700 DATA 40,5
1710 DATA AUSTRALIANA SMITH & THE FORBID
DEN TEMPLE
1720 DATA A missionary has been captured
 by the   evil witch doctor and it is up
 to you asthe intrepid explorer Australi
ana Smith to rescue her from the Forbidd
en Temple.
1730 DATA As you explore the temple you
will find many objects. Some will help y
ou in youradventure.
1740 DATA Use the commands  N (north)  S
 (south)  E (east)  W (west)  U (up)  D
(down)    T (take) & L (look).
1750 DATA Ensure that the CAPS LOCK key
is down
1799 REM LOCATION DESCRIPTION
1800 DATA You are standing at the gatewa
y to the  Land of the Forbidden Temple.
Go North  to begin your adventure.
          Hanging from the gate-post is
1810 DATA You are facing the ancient tem
ple       There is a well and row of tot
em poles  to the N. leading to the main
entrance  to the temple. There is also
1820 DATA You are to the west of the MOU
ND. On theground is
1830 DATA You are on the other side of t
he MOUND. You hear hungry voices to the
South.    On the ground is
1840 DATA You are to the east of the MOU
ND.       Hanging from a tree is
1850 DATA You are inside the entrance to
 the MOUNDThere are passages leading to
```

```
the N. E. and W. Stairs carved in the ro
ck lead UpIn a sarcophagus to one side i
s
1860 DATA A room full of cobwebs. There
is a hole to the E. In here is
1870 DATA Torture room. There is a way t
o the E.  You see
1880 DATA You are in a room with water t
rickling  down the walls. You can go E.
or W.     In here is
1890 DATA You are in a high domed chambe
r. Exits  to all sides and stairs leadin
g down.   Hanging from the roof is
1900 DATA A narrow passageway. Doors to
all sides.In a corner you see
1910 DATA The floor in here is covered w
ith dust. Exit to the W. Beneath the dus
t is
1920 DATA A small recess with a narrow p
assage to the W. On the wall to your lef
t is
1930 DATA In the centre of the room is a
 pit full of venomous snakes. You can go
 E. or W. Suspended over the pit is
1940 DATA You are at the bottom of the s
tairs.    Openings lead E. W. & S. At th
e base of the stair lies
1950 DATA A tall narrow chamber with a p
assage    leading E. In here is
1960 DATA A small room filled with stone
 jars.    Exit W. Behind the jars is
1970 DATA A short hallway with doors to
all sides.In here is
1980 DATA Skeletons hang from the walls
still heldup by thick ropes. There is a
door to   the E. You see
1990 DATA A broad low roofed cavern with
 a small  ante-chamber leads W. In here
is
2000 DATA A room dominated by a massive
stone     statue in the centre. Exit to
the E.    You see
```

```
2010 DATA A passageway with exits to the
 N. E. W. In here is
2020 DATA A crypt room. There is an exit
 to the W.and there is
2030 DATA A dead end. You can go E.
          There is
2040 DATA An ante-room. A door to the W.
 and a    tunnel leading upwards in dark
ness.     There is
2050 DATA You are at the top of the stai
rs. You   can proceed N. E. or W. You ca
n only see
2060 DATA An intersection of passages. D
oors to   all sides. Which way now??  Wh
at's this?On the floor is
2070 DATA A narrow passage openings to t
he E. W. &S.  At one end is
2080 DATA A little alcove. There is a wa
y out to  the E. There is also
2090 DATA Another alcove. This time the
door leadsW. there is
2100 DATA A room filled with rotting cor
pses.     There is a single exit to the
E.        In here is
2110 DATA More corpses. This time the ex
it is to  the W. Under a pile of filth i
s
2120 DATA You are in the altar room. You
 can smellfresh blood. You can go E. On
the altar is
2130 DATA An open cavern. There is a way
 to the W.Hidden in a crack is
2140 DATA The entrance hall. Two solid g
ates of   iron and stone block the entra
nce to theN. To one side is
2150 DATA A small village hut. You can l
eave N. orS.  In here is
2160 DATA You are at the end of a jungle
 path.    Under an overhanging root you
see
2170 DATA You are on the bank of a river
 full of  pirhanas. There is a rucksack
to one    side. You look in the rucksack
 and   see
```

```
2180 DATA You are deep into the forest u
ndergrowthIn the vines you see
2190 DATA You are at the base of a rocky
 outcrop. In a cave is
2299 REM OBJECTS
2300 DATA a large key
2310 DATA a small key
2320 DATA a brass key
2330 DATA a cloak
2340 DATA a stone vase
2350 DATA a bottle
2360 DATA a clump of moss
2370 DATA some rusted armour
2380 DATA a dead rat
2390 DATA a wooden leg
2400 DATA a mouse
2410 DATA a cat
2420 DATA a spider
2430 DATA a glass eye
2440 DATA a cup
2450 DATA a skull
2460 DATA a bell
2470 DATA a body
2480 DATA a bucket
2490 DATA a ring
2500 DATA a skeleton
2510 DATA a goblet
2520 DATA a cobweb
2530 DATA a dragon's tooth
2540 DATA a frog
2550 DATA a lamp
2560 DATA a toad
2570 DATA a broom
2580 DATA a stone
2590 DATA a club
2600 DATA a sling shot
2610 DATA a dagger
2620 DATA a spear
2630 DATA a stock whip
2640 DATA a service revolver
2650 DATA a diamond
2660 DATA a bag of coins
2670 DATA a golden skull
2680 DATA a lost ARK
```

```
2690 DATA THE MISSING MISSIONARY
2799 REM NASTIES
2800 DATA POISONOUS SNAKES
2810 DATA A GIANT SPIDER
2820 DATA A HEAD HUNTING PYGMY
2830 DATA THE WITCH DOCTORS MEN
2840 DATA THE EVIL WITCH DOCTOR
2899 REM MOVES AVAILABLE
2900 DATA 2,1,1,1,99,99,35,1,5,3,99,25,4
,2,99,37,99,99
2910 DATA 38,36,5,3,99,99,4,2,40,99,99,9
9,11,99,14,7,26,99
2920 DATA 99,99,6,99,99,99,99,99,11,99,9
9,99,99,99,10,24,99,99
2930 DATA 36,11,12,9,99,15,10,6,13,8,99,
99,99,99,99,10,99,99
2940 DATA 99,99,99,11,99,99,99,99,25,6,9
9,99,99,18,17,16,10,99
2950 DATA 99,99,15,99,99,99,99,99,99,15,
99,99,15,22,20,19,99,99
2960 DATA 99,99,18,99,99,99,99,99,99,18,
99,99,99,99,22,99,99,99
2970 DATA 18,99,23,21,99,99,99,99,99,22,
99,99,99,99,9,99,99,99
2980 DATA 99,99,99,14,2,99,27,99,34,33,9
9,6,28,26,32,31,99,99
2990 DATA 99,27,30,29,99,99,99,99,28,99,
99,99,99,99,99,28,99,99
3000 DATA 99,99,27,99,99,99,99,99,99,27,
99,99,99,99,26,99,99,99
3010 DATA 99,99,99,26,99,99,99,2,99,99,9
9,99,4,10,99,99,99,99
3020 DATA 38,1,3,99,99,99
3030 DATA 39,4,40,37,99,99
3040 DATA 99,38,99,99,99,99
3050 DATA 38,1,99,5,99,99
3060 END
```

Pitman's First Book of Amstrad Games provides an introduction to writing programs on the Amstrad series of microcomputers. There are a number of program listings covering aspects of computing ranging from quality entertainment games through to a data base program for serious application.

Each listing introduces a new basic programming technique which is explained in depth in the accompanying notes. An ideal introduction to programming in BASIC for Amstrad users.

Pitman
ISBN 0 85896 271 3